

# QuantLib

**An open source library for quantitative finance**

Version 0.3.14

Generated by Doxygen 1.5.1

2 Nov 2006



# Contents

<b>1</b>	<b>Getting started</b>	<b>1</b>
1.1	Introduction . . . . .	1
1.2	Project overview . . . . .	2
1.3	Where to get QuantLib . . . . .	10
1.4	Installation . . . . .	11
1.5	User configuration . . . . .	12
1.6	Usage . . . . .	14
1.7	Frequently asked questions . . . . .	15
1.8	Version history . . . . .	21
1.9	Additional resources . . . . .	43
1.10	The QuantLib Group . . . . .	44
1.11	QuantLib License . . . . .	45
<b>2</b>	<b>QuantLib Module Index</b>	<b>47</b>
2.1	QuantLib Modules . . . . .	47
<b>3</b>	<b>QuantLib Hierarchical Index</b>	<b>49</b>
3.1	QuantLib Class Hierarchy . . . . .	49
<b>4</b>	<b>QuantLib Class Index</b>	<b>71</b>
4.1	QuantLib Class List . . . . .	71
<b>5</b>	<b>QuantLib File Index</b>	<b>89</b>
5.1	QuantLib File List . . . . .	89
<b>6</b>	<b>QuantLib Module Documentation</b>	<b>101</b>
6.1	Numeric types . . . . .	101
6.2	Currencies and FX rates . . . . .	103
6.3	Date and time calculations . . . . .	107
6.4	Calendars . . . . .	110

6.5	Day counters . . . . .	113
6.6	Pricing engines . . . . .	114
6.7	Asian option engines . . . . .	115
6.8	Barrier option engines . . . . .	116
6.9	Basket option engines . . . . .	117
6.10	Cap/floor engines . . . . .	118
6.11	Cliquet option engines . . . . .	119
6.12	Forward option engines . . . . .	120
6.13	Quanto option engines . . . . .	121
6.14	Swaption engines . . . . .	122
6.15	Vanilla option engines . . . . .	123
6.16	Finite-differences framework . . . . .	126
6.17	Short-rate modelling framework . . . . .	128
6.18	Financial instruments . . . . .	131
6.19	Lattice methods . . . . .	135
6.20	Math tools . . . . .	138
6.21	Monte Carlo framework . . . . .	140
6.22	Design patterns . . . . .	141
6.23	Stochastic processes . . . . .	142
6.24	Term structures . . . . .	144
6.25	Utilities . . . . .	146
6.26	QuantLib macros . . . . .	148
6.27	Generic macros . . . . .	149
6.28	Numeric limits . . . . .	150
6.29	Template capabilities . . . . .	151
6.30	Iterator support . . . . .	152
6.31	Output manipulators . . . . .	153
6.32	Debugging macros . . . . .	154
<b>7</b>	<b>QuantLib Class Documentation</b>	<b>157</b>
7.1	Abcd Class Reference . . . . .	157
7.2	AbcdSquared Class Reference . . . . .	160
7.3	Actual360 Class Reference . . . . .	161
7.4	Actual365Fixed Class Reference . . . . .	162
7.5	ActualActual Class Reference . . . . .	163
7.6	AcyclicVisitor Class Reference . . . . .	164
7.7	AdditiveEQPBinoomialTree Class Reference . . . . .	165



7.8	AffineModel Class Reference . . . . .	166
7.9	AmericanCondition Class Reference . . . . .	167
7.10	AmericanExercise Class Reference . . . . .	168
7.11	AmericanPayoffAtExpiry Class Reference . . . . .	169
7.12	AmericanPayoffAtHit Class Reference . . . . .	170
7.13	AnalyticBarrierEngine Class Reference . . . . .	171
7.14	AnalyticCapFloorEngine Class Reference . . . . .	172
7.15	AnalyticCliquetEngine Class Reference . . . . .	173
7.16	AnalyticContinuousFixedLookbackEngine Class Reference . . . . .	174
7.17	AnalyticContinuousFloatingLookbackEngine Class Reference . . . . .	175
7.18	AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference . . . . .	176
7.19	AnalyticDigitalAmericanEngine Class Reference . . . . .	177
7.20	AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference . . . . .	178
7.21	AnalyticDividendEuropeanEngine Class Reference . . . . .	179
7.22	AnalyticEuropeanEngine Class Reference . . . . .	180
7.23	AnalyticHestonEngine Class Reference . . . . .	181
7.24	AnalyticPerformanceEngine Class Reference . . . . .	182
7.25	Argentina Class Reference . . . . .	183
7.26	Arguments Class Reference . . . . .	185
7.27	ArmijoLineSearch Class Reference . . . . .	186
7.28	Array Class Reference . . . . .	187
7.29	ARSCurrency Class Reference . . . . .	190
7.30	AssetOrNothingPayoff Class Reference . . . . .	191
7.31	AssetSwap Class Reference . . . . .	192
7.32	AssetSwap::arguments Class Reference . . . . .	194
7.33	AssetSwap::results Class Reference . . . . .	195
7.34	ATSCurrency Class Reference . . . . .	196
7.35	AUDCurrency Class Reference . . . . .	197
7.36	AUDLibor Class Reference . . . . .	198
7.37	Australia Class Reference . . . . .	199
7.38	Average Struct Reference . . . . .	200
7.39	BackwardFlat Class Reference . . . . .	201
7.40	BackwardFlatInterpolation Class Reference . . . . .	202
7.41	BaroneAdesiWhaleyApproximationEngine Class Reference . . . . .	203
7.42	Barrier Struct Reference . . . . .	204
7.43	BarrierOption Class Reference . . . . .	205

7.44	<a href="#">BarrierOption::arguments Class Reference</a>	207
7.45	<a href="#">BarrierOption::engine Class Reference</a>	208
7.46	<a href="#">BasketOption Class Reference</a>	209
7.47	<a href="#">BasketOption::arguments Class Reference</a>	211
7.48	<a href="#">BasketOption::engine Class Reference</a>	212
7.49	<a href="#">BatesEngine Class Reference</a>	213
7.50	<a href="#">BatesModel Class Reference</a>	215
7.51	<a href="#">BDTCurrency Class Reference</a>	216
7.52	<a href="#">BEFCurrency Class Reference</a>	217
7.53	<a href="#">BermudanExercise Class Reference</a>	218
7.54	<a href="#">BGLCurrency Class Reference</a>	219
7.55	<a href="#">Bicubic Class Reference</a>	220
7.56	<a href="#">BicubicSpline Class Reference</a>	221
7.57	<a href="#">Bilinear Class Reference</a>	222
7.58	<a href="#">BilinearInterpolation Class Reference</a>	223
7.59	<a href="#">BinomialConvertibleEngine Class Template Reference</a>	224
7.60	<a href="#">BinomialDistribution Class Reference</a>	225
7.61	<a href="#">BinomialTree Class Template Reference</a>	226
7.62	<a href="#">BinomialVanillaEngine Class Template Reference</a>	227
7.63	<a href="#">Bisection Class Reference</a>	228
7.64	<a href="#">BivariateCumulativeNormalDistributionDr78 Class Reference</a>	229
7.65	<a href="#">BivariateCumulativeNormalDistributionWe04DP Class Reference</a>	230
7.66	<a href="#">BjerkstrandStenslandApproximationEngine Class Reference</a>	231
7.67	<a href="#">BlackCapFloorEngine Class Reference</a>	232
7.68	<a href="#">BlackConstantVol Class Reference</a>	233
7.69	<a href="#">BlackFormula Class Reference</a>	235
7.70	<a href="#">BlackKarasinski Class Reference</a>	237
7.71	<a href="#">BlackKarasinski::Dynamics Class Reference</a>	238
7.72	<a href="#">BlackProcess Class Reference</a>	239
7.73	<a href="#">BlackScholesLattice Class Template Reference</a>	240
7.74	<a href="#">BlackScholesMertonProcess Class Reference</a>	241
7.75	<a href="#">BlackScholesProcess Class Reference</a>	242
7.76	<a href="#">BlackSwaptionEngine Class Reference</a>	243
7.77	<a href="#">BlackVarianceCurve Class Reference</a>	244
7.78	<a href="#">BlackVarianceSurface Class Reference</a>	246
7.79	<a href="#">BlackVarianceTermStructure Class Reference</a>	248

7.80	BlackVolatilityTermStructure Class Reference . . . . .	250
7.81	BlackVolTermStructure Class Reference . . . . .	252
7.82	Bond Class Reference . . . . .	255
7.83	BoundaryCondition Class Template Reference . . . . .	259
7.84	BoundaryConstraint Class Reference . . . . .	261
7.85	BoxMullerGaussianRng Class Template Reference . . . . .	262
7.86	Brazil Class Reference . . . . .	263
7.87	Brent Class Reference . . . . .	265
7.88	Bridge Class Template Reference . . . . .	266
7.89	BRLCurrency Class Reference . . . . .	267
7.90	BrownianBridge Class Template Reference . . . . .	268
7.91	BSMOperator Class Reference . . . . .	269
7.92	Business252 Class Reference . . . . .	270
7.93	BYRCurrency Class Reference . . . . .	271
7.94	CADCurrency Class Reference . . . . .	272
7.95	CADLibor Class Reference . . . . .	273
7.96	Calendar Class Reference . . . . .	274
7.97	Calendar::OrthodoxImpl Class Reference . . . . .	280
7.98	Calendar::WesternImpl Class Reference . . . . .	281
7.99	CalendarImpl Class Reference . . . . .	282
7.100	CalibratedModel Class Reference . . . . .	283
7.101	CalibrationHelper Class Reference . . . . .	285
7.102	Callability Class Reference . . . . .	287
7.103	Callability::Price Class Reference . . . . .	288
7.104	Canada Class Reference . . . . .	289
7.105	Cap Class Reference . . . . .	290
7.106	CapFloor Class Reference . . . . .	291
7.107	CapFloor::arguments Class Reference . . . . .	293
7.108	CapFloor::engine Class Reference . . . . .	294
7.109	CapFloor::results Class Reference . . . . .	295
7.110	CapHelper Class Reference . . . . .	296
7.111	CapletConstantVolatility Class Reference . . . . .	297
7.112	CapletVolatilityStructure Class Reference . . . . .	299
7.113	CapVolatilityStructure Class Reference . . . . .	301
7.114	CapVolatilityVector Class Reference . . . . .	303
7.115	CashFlow Class Reference . . . . .	305

7.116	Cashflows Class Reference . . . . .	307
7.117	CashOrNothingPayoff Class Reference . . . . .	310
7.118	Cdor Class Reference . . . . .	311
7.119	CeilingTruncation Class Reference . . . . .	312
7.120	CHFCurrency Class Reference . . . . .	313
7.121	CHFLibor Class Reference . . . . .	314
7.122	China Class Reference . . . . .	315
7.123	CLGaussianRng Class Template Reference . . . . .	316
7.124	CliquetOption Class Reference . . . . .	317
7.125	CliquetOption::arguments Class Reference . . . . .	319
7.126	CliquetOption::engine Class Reference . . . . .	320
7.127	Clone Class Template Reference . . . . .	321
7.128	ClosestRounding Class Reference . . . . .	322
7.129	CLPCurrency Class Reference . . . . .	323
7.130	CMSCoupon Class Reference . . . . .	324
7.131	CNYCurrency Class Reference . . . . .	326
7.132	Collar Class Reference . . . . .	327
7.133	Composite Class Template Reference . . . . .	328
7.134	CompositeConstraint Class Reference . . . . .	329
7.135	CompositeInstrument Class Reference . . . . .	330
7.136	CompositeQuote Class Template Reference . . . . .	332
7.137	CompoundForward Class Reference . . . . .	333
7.138	ConjugateGradient Class Reference . . . . .	335
7.139	ConstantEstimator Class Reference . . . . .	336
7.140	ConstantParameter Class Reference . . . . .	337
7.141	Constraint Class Reference . . . . .	338
7.142	ConstraintImpl Class Reference . . . . .	339
7.143	ContinuousAveragingAsianOption Class Reference . . . . .	340
7.144	ContinuousAveragingAsianOption::arguments Class Reference . . . . .	342
7.145	ContinuousAveragingAsianOption::engine Class Reference . . . . .	343
7.146	ContinuousFixedLookbackOption Class Reference . . . . .	344
7.147	ContinuousFixedLookbackOption::arguments Class Reference . . . . .	346
7.148	ContinuousFixedLookbackOption::engine Class Reference . . . . .	347
7.149	ContinuousFloatingLookbackOption Class Reference . . . . .	348
7.150	ContinuousFloatingLookbackOption::arguments Class Reference . . . . .	350
7.151	ContinuousFloatingLookbackOption::engine Class Reference . . . . .	351

7.152	ConundrumPricer Class Reference . . . . .	352
7.153	ConundrumPricerByNumericalIntegration Class Reference . . . . .	354
7.154	ConvergenceStatistics Class Template Reference . . . . .	355
7.155	ConvertibleBond::option::arguments Class Reference . . . . .	356
7.156	ConvertibleBond::option::engine Class Reference . . . . .	357
7.157	ConvertibleFixedCouponBond Class Reference . . . . .	358
7.158	ConvertibleFloatingRateBond Class Reference . . . . .	359
7.159	ConvertibleZeroCouponBond Class Reference . . . . .	360
7.160	COPCurrency Class Reference . . . . .	361
7.161	CostFunction Class Reference . . . . .	362
7.162	Coupon Class Reference . . . . .	363
7.163	CovarianceDecomposition Class Reference . . . . .	365
7.164	CoxIngersollRoss Class Reference . . . . .	366
7.165	CoxIngersollRoss::Dynamics Class Reference . . . . .	368
7.166	CoxRossRubinstein Class Reference . . . . .	369
7.167	CrankNicolson Class Template Reference . . . . .	370
7.168	Cubic Class Reference . . . . .	372
7.169	CubicSpline Class Reference . . . . .	373
7.170	CumulativeBinomialDistribution Class Reference . . . . .	375
7.171	CumulativeNormalDistribution Class Reference . . . . .	376
7.172	CumulativePoissonDistribution Class Reference . . . . .	377
7.173	CuriouslyRecurringTemplate Class Template Reference . . . . .	378
7.174	Currency Class Reference . . . . .	379
7.175	CurveState Class Reference . . . . .	383
7.176	CYPCurrency Class Reference . . . . .	384
7.177	CzechRepublic Class Reference . . . . .	385
7.178	CZKCurrency Class Reference . . . . .	387
7.179	Date Class Reference . . . . .	388
7.180	DayCounter Class Reference . . . . .	392
7.181	DayCounterImpl Class Reference . . . . .	394
7.182	DEMCurrency Class Reference . . . . .	395
7.183	Denmark Class Reference . . . . .	396
7.184	DepositRateHelper Class Reference . . . . .	397
7.185	DerivedQuote Class Template Reference . . . . .	398
7.186	DirichletBC Class Reference . . . . .	399
7.187	Discount Struct Reference . . . . .	400

7.188	DiscrepancyStatistics Class Reference . . . . .	401
7.189	DiscreteAveragingAsianOption Class Reference . . . . .	402
7.190	DiscreteAveragingAsianOption::arguments Class Reference . . . . .	404
7.191	DiscreteAveragingAsianOption::engine Class Reference . . . . .	405
7.192	DiscreteGeometricASO Class Reference . . . . .	406
7.193	DiscretizedAsset Class Reference . . . . .	407
7.194	DiscretizedDiscountBond Class Reference . . . . .	410
7.195	DiscretizedOption Class Reference . . . . .	411
7.196	Disposable Class Template Reference . . . . .	413
7.197	Dividend Class Reference . . . . .	414
7.198	DividendVanillaOption Class Reference . . . . .	416
7.199	DividendVanillaOption::arguments Class Reference . . . . .	418
7.200	DividendVanillaOption::engine Class Reference . . . . .	419
7.201	DKKCurrency Class Reference . . . . .	420
7.202	DKKLibor Class Reference . . . . .	421
7.203	DMinus Class Reference . . . . .	422
7.204	DownRounding Class Reference . . . . .	423
7.205	DPlus Class Reference . . . . .	424
7.206	DPlusDMinus Class Reference . . . . .	425
7.207	DriftCalculator Class Reference . . . . .	426
7.208	DriftTermStructure Class Reference . . . . .	427
7.209	Duration Struct Reference . . . . .	429
7.210	DZero Class Reference . . . . .	430
7.211	EarlyExercise Class Reference . . . . .	431
7.212	EarlyExercisePathPricer Class Template Reference . . . . .	432
7.213	EEKCurrency Class Reference . . . . .	433
7.214	EndCriteria Class Reference . . . . .	434
7.215	EqualJumpsBinomialTree Class Template Reference . . . . .	436
7.216	EqualProbabilitiesBinomialTree Class Template Reference . . . . .	437
7.217	Error Class Reference . . . . .	438
7.218	ErrorFunction Class Reference . . . . .	439
7.219	ESPCurrency Class Reference . . . . .	440
7.220	EulerDiscretization Class Reference . . . . .	441
7.221	EURCurrency Class Reference . . . . .	443
7.222	Euribor Class Reference . . . . .	444
7.223	Euribor10M Class Reference . . . . .	445

7.224	Euribor11M Class Reference . . . . .	446
7.225	Euribor1M Class Reference . . . . .	447
7.226	Euribor1Y Class Reference . . . . .	448
7.227	Euribor2M Class Reference . . . . .	449
7.228	Euribor2W Class Reference . . . . .	450
7.229	Euribor365 Class Reference . . . . .	451
7.230	Euribor365_10M Class Reference . . . . .	452
7.231	Euribor365_11M Class Reference . . . . .	453
7.232	Euribor365_1M Class Reference . . . . .	454
7.233	Euribor365_1Y Class Reference . . . . .	455
7.234	Euribor365_2M Class Reference . . . . .	456
7.235	Euribor365_2W Class Reference . . . . .	457
7.236	Euribor365_3M Class Reference . . . . .	458
7.237	Euribor365_3W Class Reference . . . . .	459
7.238	Euribor365_4M Class Reference . . . . .	460
7.239	Euribor365_5M Class Reference . . . . .	461
7.240	Euribor365_6M Class Reference . . . . .	462
7.241	Euribor365_7M Class Reference . . . . .	463
7.242	Euribor365_8M Class Reference . . . . .	464
7.243	Euribor365_9M Class Reference . . . . .	465
7.244	Euribor365_SW Class Reference . . . . .	466
7.245	Euribor3M Class Reference . . . . .	467
7.246	Euribor3W Class Reference . . . . .	468
7.247	Euribor4M Class Reference . . . . .	469
7.248	Euribor5M Class Reference . . . . .	470
7.249	Euribor6M Class Reference . . . . .	471
7.250	Euribor7M Class Reference . . . . .	472
7.251	Euribor8M Class Reference . . . . .	473
7.252	Euribor9M Class Reference . . . . .	474
7.253	EuriborSW Class Reference . . . . .	475
7.254	EuriborSwapFixA Class Reference . . . . .	476
7.255	EuriborSwapFixA10Y Class Reference . . . . .	477
7.256	EuriborSwapFixA12Y Class Reference . . . . .	478
7.257	EuriborSwapFixA15Y Class Reference . . . . .	479
7.258	EuriborSwapFixA1Y Class Reference . . . . .	480
7.259	EuriborSwapFixA20Y Class Reference . . . . .	481

7.260	EuriborSwapFixA25Y Class Reference . . . . .	482
7.261	EuriborSwapFixA2Y Class Reference . . . . .	483
7.262	EuriborSwapFixA30Y Class Reference . . . . .	484
7.263	EuriborSwapFixA3Y Class Reference . . . . .	485
7.264	EuriborSwapFixA4Y Class Reference . . . . .	486
7.265	EuriborSwapFixA5Y Class Reference . . . . .	487
7.266	EuriborSwapFixA6Y Class Reference . . . . .	488
7.267	EuriborSwapFixA7Y Class Reference . . . . .	489
7.268	EuriborSwapFixA8Y Class Reference . . . . .	490
7.269	EuriborSwapFixA9Y Class Reference . . . . .	491
7.270	EuriborSwapFixIFR Class Reference . . . . .	492
7.271	EuriborSwapFixIFR10Y Class Reference . . . . .	493
7.272	EuriborSwapFixIFR12Y Class Reference . . . . .	494
7.273	EuriborSwapFixIFR15Y Class Reference . . . . .	495
7.274	EuriborSwapFixIFR1Y Class Reference . . . . .	496
7.275	EuriborSwapFixIFR20Y Class Reference . . . . .	497
7.276	EuriborSwapFixIFR25Y Class Reference . . . . .	498
7.277	EuriborSwapFixIFR2Y Class Reference . . . . .	499
7.278	EuriborSwapFixIFR30Y Class Reference . . . . .	500
7.279	EuriborSwapFixIFR3Y Class Reference . . . . .	501
7.280	EuriborSwapFixIFR4Y Class Reference . . . . .	502
7.281	EuriborSwapFixIFR5Y Class Reference . . . . .	503
7.282	EuriborSwapFixIFR6Y Class Reference . . . . .	504
7.283	EuriborSwapFixIFR7Y Class Reference . . . . .	505
7.284	EuriborSwapFixIFR8Y Class Reference . . . . .	506
7.285	EuriborSwapFixIFR9Y Class Reference . . . . .	507
7.286	EURLibor Class Reference . . . . .	508
7.287	EURLibor10M Class Reference . . . . .	509
7.288	EURLibor11M Class Reference . . . . .	510
7.289	EURLibor1M Class Reference . . . . .	511
7.290	EURLibor1Y Class Reference . . . . .	512
7.291	EURLibor2M Class Reference . . . . .	513
7.292	EURLibor2W Class Reference . . . . .	514
7.293	EURLibor3M Class Reference . . . . .	515
7.294	EURLibor4M Class Reference . . . . .	516
7.295	EURLibor5M Class Reference . . . . .	517



7.296	EURLibor6M Class Reference . . . . .	518
7.297	EURLibor7M Class Reference . . . . .	519
7.298	EURLibor8M Class Reference . . . . .	520
7.299	EURLibor9M Class Reference . . . . .	521
7.300	EURLiborSW Class Reference . . . . .	522
7.301	EurliborSwapFixA Class Reference . . . . .	523
7.302	EurliborSwapFixA10Y Class Reference . . . . .	524
7.303	EurliborSwapFixA12Y Class Reference . . . . .	525
7.304	EurliborSwapFixA15Y Class Reference . . . . .	526
7.305	EurliborSwapFixA1Y Class Reference . . . . .	527
7.306	EurliborSwapFixA20Y Class Reference . . . . .	528
7.307	EurliborSwapFixA25Y Class Reference . . . . .	529
7.308	EurliborSwapFixA2Y Class Reference . . . . .	530
7.309	EurliborSwapFixA30Y Class Reference . . . . .	531
7.310	EurliborSwapFixA3Y Class Reference . . . . .	532
7.311	EurliborSwapFixA4Y Class Reference . . . . .	533
7.312	EurliborSwapFixA5Y Class Reference . . . . .	534
7.313	EurliborSwapFixA6Y Class Reference . . . . .	535
7.314	EurliborSwapFixA7Y Class Reference . . . . .	536
7.315	EurliborSwapFixA8Y Class Reference . . . . .	537
7.316	EurliborSwapFixA9Y Class Reference . . . . .	538
7.317	EurliborSwapFixB Class Reference . . . . .	539
7.318	EurliborSwapFixB10Y Class Reference . . . . .	540
7.319	EurliborSwapFixB12Y Class Reference . . . . .	541
7.320	EurliborSwapFixB15Y Class Reference . . . . .	542
7.321	EurliborSwapFixB1Y Class Reference . . . . .	543
7.322	EurliborSwapFixB20Y Class Reference . . . . .	544
7.323	EurliborSwapFixB25Y Class Reference . . . . .	545
7.324	EurliborSwapFixB2Y Class Reference . . . . .	546
7.325	EurliborSwapFixB30Y Class Reference . . . . .	547
7.326	EurliborSwapFixB3Y Class Reference . . . . .	548
7.327	EurliborSwapFixB4Y Class Reference . . . . .	549
7.328	EurliborSwapFixB5Y Class Reference . . . . .	550
7.329	EurliborSwapFixB6Y Class Reference . . . . .	551
7.330	EurliborSwapFixB7Y Class Reference . . . . .	552
7.331	EurliborSwapFixB8Y Class Reference . . . . .	553

7.332	EurliborSwapFixB9Y Class Reference . . . . .	554
7.333	EurliborSwapFixIFR Class Reference . . . . .	555
7.334	EurliborSwapFixIFR10Y Class Reference . . . . .	556
7.335	EurliborSwapFixIFR12Y Class Reference . . . . .	557
7.336	EurliborSwapFixIFR15Y Class Reference . . . . .	558
7.337	EurliborSwapFixIFR1Y Class Reference . . . . .	559
7.338	EurliborSwapFixIFR20Y Class Reference . . . . .	560
7.339	EurliborSwapFixIFR25Y Class Reference . . . . .	561
7.340	EurliborSwapFixIFR2Y Class Reference . . . . .	562
7.341	EurliborSwapFixIFR30Y Class Reference . . . . .	563
7.342	EurliborSwapFixIFR3Y Class Reference . . . . .	564
7.343	EurliborSwapFixIFR4Y Class Reference . . . . .	565
7.344	EurliborSwapFixIFR5Y Class Reference . . . . .	566
7.345	EurliborSwapFixIFR6Y Class Reference . . . . .	567
7.346	EurliborSwapFixIFR7Y Class Reference . . . . .	568
7.347	EurliborSwapFixIFR8Y Class Reference . . . . .	569
7.348	EurliborSwapFixIFR9Y Class Reference . . . . .	570
7.349	EuropeanExercise Class Reference . . . . .	571
7.350	EuropeanOption Class Reference . . . . .	572
7.351	Event Class Reference . . . . .	573
7.352	EvolutionDescription Class Reference . . . . .	575
7.353	ExchangeRate Class Reference . . . . .	576
7.354	ExchangeRateManager Class Reference . . . . .	578
7.355	Exercise Class Reference . . . . .	580
7.356	ExplicitEuler Class Template Reference . . . . .	581
7.357	ExtendedCoxIngersollRoss Class Reference . . . . .	583
7.358	ExtendedCoxIngersollRoss::Dynamics Class Reference . . . . .	585
7.359	ExtendedCoxIngersollRoss::FittingParameter Class Reference . . . . .	586
7.360	ExtendedDiscountCurve Class Reference . . . . .	587
7.361	Extrapolator Class Reference . . . . .	589
7.362	Factorial Class Reference . . . . .	590
7.363	FalsePosition Class Reference . . . . .	591
7.364	FaureRsg Class Reference . . . . .	592
7.365	FDAmericanCondition Class Template Reference . . . . .	593
7.366	FDBermudanEngine Class Reference . . . . .	594
7.367	FDDividendEngineMerton73 Class Reference . . . . .	595

7.368	FDDividendEngineShiftScale Class Reference . . . . .	596
7.369	FDEuropeanEngine Class Reference . . . . .	597
7.370	FDStepConditionEngine Class Reference . . . . .	598
7.371	FIMCurrency Class Reference . . . . .	599
7.372	FiniteDifferenceModel Class Template Reference . . . . .	600
7.373	Finland Class Reference . . . . .	601
7.374	FixedCouponBond Class Reference . . . . .	602
7.375	FixedCouponBondForward Class Reference . . . . .	604
7.376	FixedCouponBondHelper Class Reference . . . . .	607
7.377	FixedDividend Class Reference . . . . .	609
7.378	FixedRateCoupon Class Reference . . . . .	611
7.379	FlatForward Class Reference . . . . .	613
7.380	FloatingRateBond Class Reference . . . . .	615
7.381	FloatingRateCoupon Class Reference . . . . .	617
7.382	FloatingTypePayoff Class Reference . . . . .	620
7.383	Floor Class Reference . . . . .	621
7.384	FloorTruncation Class Reference . . . . .	622
7.385	Forward Class Reference . . . . .	623
7.386	ForwardEngine Class Template Reference . . . . .	626
7.387	ForwardFlat Class Reference . . . . .	627
7.388	ForwardFlatInterpolation Class Reference . . . . .	628
7.389	ForwardMeasureProcess Class Reference . . . . .	629
7.390	ForwardMeasureProcess1D Class Reference . . . . .	630
7.391	ForwardOptionArguments Class Template Reference . . . . .	631
7.392	ForwardPerformanceEngine Class Template Reference . . . . .	632
7.393	ForwardRate Struct Reference . . . . .	633
7.394	ForwardRateAgreement Class Reference . . . . .	634
7.395	ForwardRateIpcEvolver Class Reference . . . . .	637
7.396	ForwardRatePcEvolver Class Reference . . . . .	638
7.397	ForwardRateStructure Class Reference . . . . .	639
7.398	ForwardSpreadedTermStructure Class Reference . . . . .	641
7.399	ForwardTypePayoff Class Reference . . . . .	643
7.400	ForwardVanillaOption Class Reference . . . . .	644
7.401	FractionalDividend Class Reference . . . . .	646
7.402	FraRateHelper Class Reference . . . . .	648
7.403	FRFCurrency Class Reference . . . . .	649

7.404	FuturesRateHelper Class Reference . . . . .	650
7.405	G2 Class Reference . . . . .	651
7.406	G2::FittingParameter Class Reference . . . . .	653
7.407	G2ForwardProcess Class Reference . . . . .	654
7.408	G2Process Class Reference . . . . .	656
7.409	G2SwaptionEngine Class Reference . . . . .	658
7.410	GammaFunction Class Reference . . . . .	659
7.411	GapPayoff Class Reference . . . . .	660
7.412	Garch11 Class Reference . . . . .	661
7.413	GarmanKlassAbstract Class Reference . . . . .	662
7.414	GarmanKlassOpenClose Class Template Reference . . . . .	663
7.415	GarmanKohlagenProcess Class Reference . . . . .	664
7.416	GaussChebyshev2thIntegration Class Reference . . . . .	665
7.417	GaussChebyshevIntegration Class Reference . . . . .	666
7.418	GaussGegenbauerIntegration Class Reference . . . . .	667
7.419	GaussHermiteIntegration Class Reference . . . . .	668
7.420	GaussHermitePolynomial Class Reference . . . . .	669
7.421	GaussHyperbolicIntegration Class Reference . . . . .	670
7.422	GaussHyperbolicPolynomial Class Reference . . . . .	671
7.423	GaussianOrthogonalPolynomial Class Reference . . . . .	672
7.424	GaussianQuadrature Class Reference . . . . .	673
7.425	GaussJacobiIntegration Class Reference . . . . .	674
7.426	GaussJacobiPolynomial Class Reference . . . . .	675
7.427	GaussLaguerreIntegration Class Reference . . . . .	676
7.428	GaussLaguerrePolynomial Class Reference . . . . .	677
7.429	GaussLegendreIntegration Class Reference . . . . .	678
7.430	GBPCurrency Class Reference . . . . .	679
7.431	GBPLibor Class Reference . . . . .	680
7.432	GeneralizedBlackScholesProcess Class Reference . . . . .	681
7.433	GeneralStatistics Class Reference . . . . .	683
7.434	GenericEngine Class Template Reference . . . . .	686
7.435	GenericGaussianStatistics Class Template Reference . . . . .	688
7.436	GenericModelEngine Class Template Reference . . . . .	691
7.437	GenericRiskStatistics Class Template Reference . . . . .	692
7.438	GenericSequenceStatistics Class Template Reference . . . . .	695
7.439	GeometricBrownianMotionProcess Class Reference . . . . .	697

7.440	Germany Class Reference . . . . .	698
7.441	GRDCurrency Class Reference . . . . .	701
7.442	Greeks Class Reference . . . . .	702
7.443	HaltonRsg Class Reference . . . . .	703
7.444	Handle Class Template Reference . . . . .	704
7.445	HestonModel Class Reference . . . . .	706
7.446	HestonModelHelper Class Reference . . . . .	707
7.447	HestonProcess Class Reference . . . . .	708
7.448	HKDCurrency Class Reference . . . . .	710
7.449	HongKong Class Reference . . . . .	711
7.450	HUFCurrency Class Reference . . . . .	713
7.451	HullWhite Class Reference . . . . .	714
7.452	HullWhite::Dynamics Class Reference . . . . .	716
7.453	HullWhite::FittingParameter Class Reference . . . . .	717
7.454	HullWhiteForwardProcess Class Reference . . . . .	718
7.455	HullWhiteProcess Class Reference . . . . .	720
7.456	Hungary Class Reference . . . . .	722
7.457	Iceland Class Reference . . . . .	723
7.458	IEPCurrency Class Reference . . . . .	725
7.459	ILSCurrency Class Reference . . . . .	726
7.460	IMM Struct Reference . . . . .	727
7.461	ImplicitEuler Class Template Reference . . . . .	728
7.462	ImpliedTermStructure Class Reference . . . . .	729
7.463	ImpliedVolTermStructure Class Reference . . . . .	731
7.464	InArrearIndexedCoupon Class Reference . . . . .	733
7.465	IncrementalStatistics Class Reference . . . . .	735
7.466	Index Class Reference . . . . .	738
7.467	IndexManager Class Reference . . . . .	740
7.468	India Class Reference . . . . .	741
7.469	Indonesia Class Reference . . . . .	743
7.470	INRCurrency Class Reference . . . . .	745
7.471	Instrument Class Reference . . . . .	746
7.472	IntegralEngine Class Reference . . . . .	749
7.473	InterestRate Class Reference . . . . .	750
7.474	InterestRateIndex Class Reference . . . . .	753
7.475	InterpolatedDiscountCurve Class Template Reference . . . . .	755

7.476	InterpolatedForwardCurve Class Template Reference . . . . .	757
7.477	InterpolatedZeroCurve Class Template Reference . . . . .	759
7.478	Interpolation Class Reference . . . . .	761
7.479	Interpolation2D Class Reference . . . . .	762
7.480	Interpolation2D::templateImpl Class Template Reference . . . . .	764
7.481	Interpolation2DImpl Class Reference . . . . .	765
7.482	Interpolation::templateImpl Class Template Reference . . . . .	766
7.483	InterpolationImpl Class Reference . . . . .	767
7.484	IntervalPrice Class Reference . . . . .	768
7.485	InverseCumulativeNormal Class Reference . . . . .	769
7.486	InverseCumulativePoisson Class Reference . . . . .	770
7.487	InverseCumulativeRng Class Template Reference . . . . .	771
7.488	InverseCumulativeRsg Class Template Reference . . . . .	772
7.489	IQDCurrency Class Reference . . . . .	773
7.490	IRRCurrency Class Reference . . . . .	774
7.491	ISKCurrency Class Reference . . . . .	775
7.492	Italy Class Reference . . . . .	776
7.493	ITLCurrency Class Reference . . . . .	778
7.494	JamshidianSwaptionEngine Class Reference . . . . .	779
7.495	Japan Class Reference . . . . .	780
7.496	JarrowRudd Class Reference . . . . .	782
7.497	Jibar Class Reference . . . . .	783
7.498	JointCalendar Class Reference . . . . .	784
7.499	JPYCurrency Class Reference . . . . .	785
7.500	JPYLibor Class Reference . . . . .	786
7.501	JumpDiffusionEngine Class Reference . . . . .	787
7.502	JuQuadraticApproximationEngine Class Reference . . . . .	788
7.503	KnuthUniformRng Class Reference . . . . .	789
7.504	KronrodIntegral Class Reference . . . . .	790
7.505	KRWCurrency Class Reference . . . . .	791
7.506	KWDCurrency Class Reference . . . . .	792
7.507	Lattice Class Template Reference . . . . .	793
7.508	Lattice1D Class Template Reference . . . . .	795
7.509	Lattice2D Class Template Reference . . . . .	796
7.510	LatticeShortRateModelEngine Class Template Reference . . . . .	797
7.511	LazyObject Class Reference . . . . .	798

7.512	LeastSquareFunction Class Reference . . . . .	800
7.513	LeastSquareProblem Class Reference . . . . .	801
7.514	LecuyerUniformRng Class Reference . . . . .	802
7.515	LeisenReimer Class Reference . . . . .	803
7.516	LevenbergMarquardt Class Reference . . . . .	804
7.517	LexicographicalView Class Template Reference . . . . .	805
7.518	LfmCovarianceParameterization Class Reference . . . . .	807
7.519	LfmCovarianceProxy Class Reference . . . . .	808
7.520	LfmHullWhiteParameterization Class Reference . . . . .	809
7.521	LfmSwaptionEngine Class Reference . . . . .	810
7.522	Libor Class Reference . . . . .	811
7.523	LiborForwardModel Class Reference . . . . .	812
7.524	LiborForwardModelProcess Class Reference . . . . .	814
7.525	Linear Class Reference . . . . .	816
7.526	LinearInterpolation Class Reference . . . . .	817
7.527	LinearLeastSquaresRegression Class Template Reference . . . . .	818
7.528	LineSearch Class Reference . . . . .	819
7.529	Link Class Template Reference . . . . .	821
7.530	LmConstWrapperVolatilityModel Class Reference . . . . .	823
7.531	LmCorrelationModel Class Reference . . . . .	824
7.532	LmExponentialCorrelationModel Class Reference . . . . .	825
7.533	LmExtLinearExponentialVolModel Class Reference . . . . .	826
7.534	LmLinearExponentialCorrelationModel Class Reference . . . . .	827
7.535	LmLinearExponentialVolatilityModel Class Reference . . . . .	828
7.536	LmVolatilityModel Class Reference . . . . .	829
7.537	LocalConstantVol Class Reference . . . . .	830
7.538	LocalVolatilityEstimator Class Template Reference . . . . .	832
7.539	LocalVolCurve Class Reference . . . . .	833
7.540	LocalVolSurface Class Reference . . . . .	835
7.541	LocalVolTermStructure Class Reference . . . . .	837
7.542	LogLinear Class Reference . . . . .	839
7.543	LogLinearInterpolation Class Reference . . . . .	840
7.544	LongstaffSchwartzPathPricer Class Template Reference . . . . .	841
7.545	LTLCurrency Class Reference . . . . .	842
7.546	LUFCurrency Class Reference . . . . .	843
7.547	LVLCurrency Class Reference . . . . .	844

7.548	MakeMCAmericanEngine Class Template Reference . . . . .	845
7.549	MakeMCDigitalEngine Class Template Reference . . . . .	846
7.550	MakeMCEuropeanEngine Class Template Reference . . . . .	847
7.551	MakeMCEuropeanHestonEngine Class Template Reference . . . . .	848
7.552	MakeMCHullWhiteCapFloorEngine Class Template Reference . . . . .	849
7.553	MakeMCVarianceSwapEngine Class Template Reference . . . . .	850
7.554	MakeSchedule Class Reference . . . . .	851
7.555	MakeVanillaSwap Class Reference . . . . .	852
7.556	MarketModelComposite Class Reference . . . . .	853
7.557	MarketModelEvolver Class Reference . . . . .	855
7.558	MarketModelMultiProduct Class Reference . . . . .	856
7.559	Matrix Class Reference . . . . .	857
7.560	MCAmericanBasketEngine Class Template Reference . . . . .	861
7.561	MCAmericanEngine Class Template Reference . . . . .	862
7.562	MCBarrierEngine Class Template Reference . . . . .	863
7.563	MCBasketEngine Class Template Reference . . . . .	865
7.564	McCliquetOption Class Reference . . . . .	867
7.565	MCDigitalEngine Class Template Reference . . . . .	868
7.566	MCDiscreteArithmeticAPEngine Class Template Reference . . . . .	869
7.567	MCDiscreteArithmeticASO Class Reference . . . . .	870
7.568	MCDiscreteAveragingAsianEngine Class Template Reference . . . . .	871
7.569	MCDiscreteGeometricAPEngine Class Template Reference . . . . .	873
7.570	MCEuropeanEngine Class Template Reference . . . . .	874
7.571	MCEuropeanHestonEngine Class Template Reference . . . . .	875
7.572	McEverest Class Reference . . . . .	876
7.573	McHimalaya Class Reference . . . . .	877
7.574	MCHullWhiteCapFloorEngine Class Template Reference . . . . .	878
7.575	MCLongstaffSchwartzEngine Class Template Reference . . . . .	879
7.576	McMaxBasket Class Reference . . . . .	881
7.577	McPagoda Class Reference . . . . .	882
7.578	McPerformanceOption Class Reference . . . . .	883
7.579	McPricer Class Template Reference . . . . .	884
7.580	McSimulation Class Template Reference . . . . .	885
7.581	MCVanillaEngine Class Template Reference . . . . .	887
7.582	MCVarianceSwapEngine Class Template Reference . . . . .	888
7.583	MersenneTwisterUniformRng Class Reference . . . . .	890



7.584	Merton76Process Class Reference . . . . .	891
7.585	Mexico Class Reference . . . . .	893
7.586	MixedScheme Class Template Reference . . . . .	895
7.587	Money Class Reference . . . . .	897
7.588	MonotonicCubicSpline Class Reference . . . . .	899
7.589	MonteCarloModel Class Template Reference . . . . .	900
7.590	MoreGreeks Class Reference . . . . .	901
7.591	MoroInverseCumulativeNormal Class Reference . . . . .	902
7.592	MTBrownianGenerator Class Reference . . . . .	903
7.593	MTLCurrency Class Reference . . . . .	904
7.594	MultiAssetOption Class Reference . . . . .	905
7.595	MultiAssetOption::arguments Class Reference . . . . .	907
7.596	MultiAssetOption::results Class Reference . . . . .	908
7.597	MultiCubicSpline Class Template Reference . . . . .	909
7.598	MultiPath Class Reference . . . . .	910
7.599	MultiPathGenerator Class Template Reference . . . . .	911
7.600	MultiProductComposite Class Reference . . . . .	912
7.601	MultiProductMultiStep Class Reference . . . . .	913
7.602	MultiProductOneStep Class Reference . . . . .	914
7.603	MultiVariate Struct Template Reference . . . . .	915
7.604	MXNCurrency Class Reference . . . . .	916
7.605	NaturalCubicSpline Class Reference . . . . .	917
7.606	NaturalMonotonicCubicSpline Class Reference . . . . .	918
7.607	NeumannBC Class Reference . . . . .	919
7.608	Newton Class Reference . . . . .	920
7.609	NewtonSafe Class Reference . . . . .	921
7.610	NewZealand Class Reference . . . . .	922
7.611	NLGCurrency Class Reference . . . . .	923
7.612	NoConstraint Class Reference . . . . .	924
7.613	NOKCurrency Class Reference . . . . .	925
7.614	NonLinearLeastSquare Class Reference . . . . .	926
7.615	NormalDistribution Class Reference . . . . .	927
7.616	Norway Class Reference . . . . .	928
7.617	NPRCurrency Class Reference . . . . .	929
7.618	Null Class Template Reference . . . . .	930
7.619	NullCalendar Class Reference . . . . .	931

7.620	NullCondition Class Template Reference . . . . .	932
7.621	NullParameter Class Reference . . . . .	933
7.622	NumericalMethod Class Reference . . . . .	934
7.623	NZDCurrency Class Reference . . . . .	936
7.624	NZDLibor Class Reference . . . . .	937
7.625	Observable Class Reference . . . . .	938
7.626	ObservableValue Class Template Reference . . . . .	940
7.627	Observer Class Reference . . . . .	941
7.628	OneAssetOption Class Reference . . . . .	943
7.629	OneAssetOption::arguments Class Reference . . . . .	946
7.630	OneAssetOption::results Class Reference . . . . .	947
7.631	OneAssetStrikedOption Class Reference . . . . .	948
7.632	OneDayCounter Class Reference . . . . .	950
7.633	OneFactorAffineModel Class Reference . . . . .	951
7.634	OneFactorModel Class Reference . . . . .	952
7.635	OneFactorModel::ShortRateDynamics Class Reference . . . . .	953
7.636	OneFactorModel::ShortRateTree Class Reference . . . . .	954
7.637	OperatorFactory Class Reference . . . . .	955
7.638	OptimizationMethod Class Reference . . . . .	956
7.639	Option Class Reference . . . . .	958
7.640	Option::arguments Class Reference . . . . .	959
7.641	OrnsteinUhlenbeckProcess Class Reference . . . . .	960
7.642	Parameter Class Reference . . . . .	962
7.643	ParameterImpl Class Reference . . . . .	963
7.644	ParCoupon Class Reference . . . . .	964
7.645	Path Class Reference . . . . .	965
7.646	PathGenerator Class Template Reference . . . . .	966
7.647	PathPricer Class Template Reference . . . . .	967
7.648	Payoff Class Reference . . . . .	968
7.649	PercentageStrikePayoff Class Reference . . . . .	969
7.650	Period Class Reference . . . . .	970
7.651	PiecewiseConstantParameter Class Reference . . . . .	971
7.652	PiecewiseYieldCurve Class Template Reference . . . . .	972
7.653	PiecewiseZeroSpreadedTermStructure Class Reference . . . . .	974
7.654	PKRCurrency Class Reference . . . . .	976
7.655	PlainVanillaPayoff Class Reference . . . . .	977

7.656	PLNCurrency Class Reference . . . . .	978
7.657	PoissonDistribution Class Reference . . . . .	979
7.658	Poland Class Reference . . . . .	980
7.659	PositiveConstraint Class Reference . . . . .	981
7.660	PriceCurve Class Reference . . . . .	982
7.661	PricingEngine Class Reference . . . . .	983
7.662	PrimeNumbers Class Reference . . . . .	984
7.663	Problem Class Reference . . . . .	985
7.664	PTECurrency Class Reference . . . . .	987
7.665	QuantoEngine Class Template Reference . . . . .	988
7.666	QuantoForwardVanillaOption Class Reference . . . . .	990
7.667	QuantoOptionArguments Class Template Reference . . . . .	992
7.668	QuantoOptionResults Class Template Reference . . . . .	993
7.669	QuantoTermStructure Class Reference . . . . .	994
7.670	QuantoVanillaOption Class Reference . . . . .	996
7.671	Quote Class Reference . . . . .	998
7.672	RandomizedLDS Class Template Reference . . . . .	999
7.673	RandomSequenceGenerator Class Template Reference . . . . .	1001
7.674	RateHelper Class Reference . . . . .	1002
7.675	RelativeDateRateHelper Class Reference . . . . .	1004
7.676	ReplicatingVarianceSwapEngine Class Reference . . . . .	1005
7.677	Results Class Reference . . . . .	1006
7.678	Ridder Class Reference . . . . .	1007
7.679	ROLCurrency Class Reference . . . . .	1008
7.680	RONCurrency Class Reference . . . . .	1009
7.681	Rounding Class Reference . . . . .	1010
7.682	SABRInterpolation Class Reference . . . . .	1012
7.683	SalvagingAlgorithm Struct Reference . . . . .	1013
7.684	Sample Struct Template Reference . . . . .	1014
7.685	SampledCurve Class Reference . . . . .	1015
7.686	SARCurrency Class Reference . . . . .	1017
7.687	SaudiArabia Class Reference . . . . .	1018
7.688	Schedule Class Reference . . . . .	1019
7.689	Secant Class Reference . . . . .	1021
7.690	SeedGenerator Class Reference . . . . .	1022
7.691	SegmentIntegral Class Reference . . . . .	1023

7.692	SEKCurrency Class Reference . . . . .	1024
7.693	Settings Class Reference . . . . .	1025
7.694	Settlement Struct Reference . . . . .	1027
7.695	SGDCurrency Class Reference . . . . .	1028
7.696	Short Class Template Reference . . . . .	1029
7.697	Short< ParCoupon > Class Template Reference . . . . .	1030
7.698	ShortRateModel Class Reference . . . . .	1031
7.699	ShoutCondition Class Reference . . . . .	1032
7.700	SimpleCashFlow Class Reference . . . . .	1033
7.701	SimpleDayCounter Class Reference . . . . .	1034
7.702	SimpleLocalEstimator Class Reference . . . . .	1035
7.703	SimpleQuote Class Reference . . . . .	1036
7.704	Simplex Class Reference . . . . .	1037
7.705	SimpsonIntegral Class Reference . . . . .	1038
7.706	Singapore Class Reference . . . . .	1039
7.707	SingleAssetOption Class Reference . . . . .	1041
7.708	SingleProductComposite Class Reference . . . . .	1043
7.709	Singleton Class Template Reference . . . . .	1044
7.710	SingleVariate Struct Template Reference . . . . .	1045
7.711	SITCurrency Class Reference . . . . .	1046
7.712	SKKCurrency Class Reference . . . . .	1047
7.713	Slovakia Class Reference . . . . .	1048
7.714	SmileSection Class Reference . . . . .	1050
7.715	SobolRsg Class Reference . . . . .	1051
7.716	SoftCallability Class Reference . . . . .	1053
7.717	Solver1D Class Template Reference . . . . .	1054
7.718	SouthAfrica Class Reference . . . . .	1056
7.719	SouthKorea Class Reference . . . . .	1057
7.720	SquareRootProcess Class Reference . . . . .	1059
7.721	StatsHolder Class Reference . . . . .	1060
7.722	SteepestDescent Class Reference . . . . .	1061
7.723	step_iterator Class Template Reference . . . . .	1062
7.724	StepCondition Class Template Reference . . . . .	1063
7.725	StepConditionSet Class Template Reference . . . . .	1064
7.726	StochasticProcess Class Reference . . . . .	1065
7.727	StochasticProcess1D Class Reference . . . . .	1068

7.728	StochasticProcess1D::discretization Class Reference . . . . .	1070
7.729	StochasticProcess::discretization Class Reference . . . . .	1071
7.730	StochasticProcessArray Class Reference . . . . .	1072
7.731	Stock Class Reference . . . . .	1074
7.732	StrikedTypePayoff Class Reference . . . . .	1075
7.733	StulzEngine Class Reference . . . . .	1076
7.734	SuperSharePayoff Class Reference . . . . .	1077
7.735	SVD Class Reference . . . . .	1078
7.736	Swap Class Reference . . . . .	1079
7.737	SwapIndex Class Reference . . . . .	1081
7.738	SwapRateHelper Class Reference . . . . .	1083
7.739	Swaption Class Reference . . . . .	1085
7.740	Swaption::arguments Class Reference . . . . .	1087
7.741	Swaption::engine Class Reference . . . . .	1088
7.742	Swaption::results Class Reference . . . . .	1089
7.743	SwaptionConstantVolatility Class Reference . . . . .	1090
7.744	SwaptionHelper Class Reference . . . . .	1092
7.745	SwaptionVolatilityCube Class Reference . . . . .	1093
7.746	SwaptionVolatilityCubeBySabr Class Reference . . . . .	1095
7.747	SwaptionVolatilityMatrix Class Reference . . . . .	1097
7.748	SwaptionVolatilityStructure Class Reference . . . . .	1099
7.749	Sweden Class Reference . . . . .	1102
7.750	Switzerland Class Reference . . . . .	1103
7.751	SymmetricSchurDecomposition Class Reference . . . . .	1104
7.752	TabulatedGaussLegendre Class Reference . . . . .	1105
7.753	Taiwan Class Reference . . . . .	1106
7.754	TARGET Class Reference . . . . .	1108
7.755	TermStructure Class Reference . . . . .	1109
7.756	TermStructureConsistentModel Class Reference . . . . .	1111
7.757	TermStructureFittingParameter Class Reference . . . . .	1112
7.758	THBCurrency Class Reference . . . . .	1113
7.759	Thirty360 Class Reference . . . . .	1114
7.760	Tian Class Reference . . . . .	1115
7.761	Tibor Class Reference . . . . .	1116
7.762	TimeBasket Class Reference . . . . .	1117
7.763	TimeGrid Class Reference . . . . .	1118

7.764	TimeSeries Class Template Reference . . . . .	1120
7.765	TqrEigenDecomposition Class Reference . . . . .	1122
7.766	TransformedGrid Class Reference . . . . .	1123
7.767	TrapezoidIntegral Class Reference . . . . .	1124
7.768	Tree Class Template Reference . . . . .	1126
7.769	TreeCapFloorEngine Class Reference . . . . .	1127
7.770	TreeSwaptionEngine Class Reference . . . . .	1128
7.771	TreeVanillaSwapEngine Class Reference . . . . .	1129
7.772	TridiagonalOperator Class Reference . . . . .	1130
7.773	TridiagonalOperator::TimeSetter Class Reference . . . . .	1132
7.774	Trigeorgis Class Reference . . . . .	1133
7.775	TrinomialTree Class Reference . . . . .	1134
7.776	TRLCurrency Class Reference . . . . .	1135
7.777	TRLibor Class Reference . . . . .	1136
7.778	TRYCurrency Class Reference . . . . .	1137
7.779	TsiveriotisFernandesLattice Class Template Reference . . . . .	1138
7.780	TTDCurrency Class Reference . . . . .	1140
7.781	Turkey Class Reference . . . . .	1141
7.782	TWDCurrency Class Reference . . . . .	1142
7.783	TwoFactorModel Class Reference . . . . .	1143
7.784	TwoFactorModel::ShortRateDynamics Class Reference . . . . .	1144
7.785	TwoFactorModel::ShortRateTree Class Reference . . . . .	1145
7.786	TypePayoff Class Reference . . . . .	1146
7.787	Ukraine Class Reference . . . . .	1147
7.788	UnitedKingdom Class Reference . . . . .	1149
7.789	UnitedStates Class Reference . . . . .	1151
7.790	UpFrontIndexedCoupon Class Reference . . . . .	1154
7.791	UpRounding Class Reference . . . . .	1155
7.792	USDCurrency Class Reference . . . . .	1156
7.793	USDLibor Class Reference . . . . .	1157
7.794	Value Class Reference . . . . .	1158
7.795	VanillaCMSCouponPricer Class Reference . . . . .	1159
7.796	VanillaOption Class Reference . . . . .	1160
7.797	VanillaOption::engine Class Reference . . . . .	1161
7.798	VanillaSwap Class Reference . . . . .	1162
7.799	VanillaSwap::arguments Class Reference . . . . .	1164

7.800	VanillaSwap::results Class Reference . . . . .	1165
7.801	VarianceSwap Class Reference . . . . .	1166
7.802	VarianceSwap::arguments Class Reference . . . . .	1169
7.803	VarianceSwap::engine Class Reference . . . . .	1170
7.804	VarianceSwap::results Class Reference . . . . .	1171
7.805	Vasicek Class Reference . . . . .	1172
7.806	Vasicek::Dynamics Class Reference . . . . .	1174
7.807	VEBCurrency Class Reference . . . . .	1175
7.808	Visitor Class Template Reference . . . . .	1176
7.809	Xibor Class Reference . . . . .	1177
7.810	YieldTermStructure Class Reference . . . . .	1179
7.811	ZARCurrency Class Reference . . . . .	1183
7.812	ZeroCondition Class Template Reference . . . . .	1184
7.813	ZeroCouponBond Class Reference . . . . .	1185
7.814	ZeroSpreadedTermStructure Class Reference . . . . .	1186
7.815	ZeroYield Struct Reference . . . . .	1188
7.816	ZeroYieldStructure Class Reference . . . . .	1189
7.817	Zibor Class Reference . . . . .	1190
<b>8</b>	<b>QuantLib File Documentation</b>	<b>1191</b>
8.1	ql/argsandresults.hpp File Reference . . . . .	1191
8.2	ql/calendar.hpp File Reference . . . . .	1193
8.3	ql/Calendars/argentina.hpp File Reference . . . . .	1195
8.4	ql/Calendars/australia.hpp File Reference . . . . .	1196
8.5	ql/Calendars/brazil.hpp File Reference . . . . .	1197
8.6	ql/Calendars/canada.hpp File Reference . . . . .	1198
8.7	ql/Calendars/china.hpp File Reference . . . . .	1199
8.8	ql/Calendars/czechrepublic.hpp File Reference . . . . .	1200
8.9	ql/Calendars/denmark.hpp File Reference . . . . .	1201
8.10	ql/Calendars/finland.hpp File Reference . . . . .	1202
8.11	ql/Calendars/germany.hpp File Reference . . . . .	1203
8.12	ql/Calendars/hongkong.hpp File Reference . . . . .	1204
8.13	ql/Calendars/hungary.hpp File Reference . . . . .	1205
8.14	ql/Calendars/iceland.hpp File Reference . . . . .	1206
8.15	ql/Calendars/india.hpp File Reference . . . . .	1207
8.16	ql/Calendars/indonesia.hpp File Reference . . . . .	1208
8.17	ql/Calendars/italy.hpp File Reference . . . . .	1209

8.18	ql/Calendars/japan.hpp File Reference . . . . .	1210
8.19	ql/Calendars/jointcalendar.hpp File Reference . . . . .	1211
8.20	ql/Calendars/mexico.hpp File Reference . . . . .	1212
8.21	ql/Calendars/newzealand.hpp File Reference . . . . .	1213
8.22	ql/Calendars/norway.hpp File Reference . . . . .	1214
8.23	ql/Calendars/nullcalendar.hpp File Reference . . . . .	1215
8.24	ql/Calendars/poland.hpp File Reference . . . . .	1216
8.25	ql/Calendars/saudiArabia.hpp File Reference . . . . .	1217
8.26	ql/Calendars/singapore.hpp File Reference . . . . .	1218
8.27	ql/Calendars/slovakia.hpp File Reference . . . . .	1219
8.28	ql/Calendars/southAfrica.hpp File Reference . . . . .	1220
8.29	ql/Calendars/southKorea.hpp File Reference . . . . .	1221
8.30	ql/Calendars/sweden.hpp File Reference . . . . .	1222
8.31	ql/Calendars/switzerland.hpp File Reference . . . . .	1223
8.32	ql/Calendars/taiwan.hpp File Reference . . . . .	1224
8.33	ql/Calendars/target.hpp File Reference . . . . .	1225
8.34	ql/Calendars/turkey.hpp File Reference . . . . .	1226
8.35	ql/Calendars/ukraine.hpp File Reference . . . . .	1227
8.36	ql/Calendars/unitedKingdom.hpp File Reference . . . . .	1228
8.37	ql/Calendars/unitedStates.hpp File Reference . . . . .	1229
8.38	ql/capvolstructures.hpp File Reference . . . . .	1230
8.39	ql/cashflow.hpp File Reference . . . . .	1231
8.40	ql/CashFlows/analysis.hpp File Reference . . . . .	1232
8.41	ql/CashFlows/cashflowvectors.hpp File Reference . . . . .	1233
8.42	ql/CashFlows/cmscoupon.hpp File Reference . . . . .	1234
8.43	ql/CashFlows/conundrumpricer.hpp File Reference . . . . .	1236
8.44	ql/CashFlows/coupon.hpp File Reference . . . . .	1237
8.45	ql/CashFlows/dividend.hpp File Reference . . . . .	1238
8.46	ql/CashFlows/fixRateCoupon.hpp File Reference . . . . .	1239
8.47	ql/CashFlows/floatingRateCoupon.hpp File Reference . . . . .	1240
8.48	ql/CashFlows/inArrearIndexedCoupon.hpp File Reference . . . . .	1241
8.49	ql/CashFlows/indexedCashFlowvectors.hpp File Reference . . . . .	1242
8.50	ql/CashFlows/indexedCoupon.hpp File Reference . . . . .	1243
8.51	ql/CashFlows/parcoupon.hpp File Reference . . . . .	1244
8.52	ql/CashFlows/shortFloatingCoupon.hpp File Reference . . . . .	1245
8.53	ql/CashFlows/shortIndexedCoupon.hpp File Reference . . . . .	1246



8.54	ql/CashFlows/simplecashflow.hpp File Reference . . . . .	1247
8.55	ql/CashFlows/timebasket.hpp File Reference . . . . .	1248
8.56	ql/CashFlows/upfrontindexedcoupon.hpp File Reference . . . . .	1249
8.57	ql/Currencies/africa.hpp File Reference . . . . .	1250
8.58	ql/Currencies/america.hpp File Reference . . . . .	1251
8.59	ql/Currencies/asia.hpp File Reference . . . . .	1252
8.60	ql/Currencies/europe.hpp File Reference . . . . .	1254
8.61	ql/Currencies/exchangeratemanager.hpp File Reference . . . . .	1257
8.62	ql/Currencies/oceania.hpp File Reference . . . . .	1258
8.63	ql/currency.hpp File Reference . . . . .	1259
8.64	ql/date.hpp File Reference . . . . .	1260
8.65	ql/daycounter.hpp File Reference . . . . .	1263
8.66	ql/DayCounters/actual360.hpp File Reference . . . . .	1264
8.67	ql/DayCounters/actual365fixed.hpp File Reference . . . . .	1265
8.68	ql/DayCounters/actualactual.hpp File Reference . . . . .	1266
8.69	ql/DayCounters/business252.hpp File Reference . . . . .	1267
8.70	ql/DayCounters/one.hpp File Reference . . . . .	1268
8.71	ql/DayCounters/simpliedaycounter.hpp File Reference . . . . .	1269
8.72	ql/DayCounters/thirty360.hpp File Reference . . . . .	1270
8.73	ql/discretizedasset.hpp File Reference . . . . .	1271
8.74	ql/errors.hpp File Reference . . . . .	1272
8.75	ql/event.hpp File Reference . . . . .	1275
8.76	ql/exchangerate.hpp File Reference . . . . .	1276
8.77	ql/exercise.hpp File Reference . . . . .	1277
8.78	ql/FiniteDifferences/americancondition.hpp File Reference . . . . .	1278
8.79	ql/FiniteDifferences/boundarycondition.hpp File Reference . . . . .	1279
8.80	ql/FiniteDifferences/bsmoperator.hpp File Reference . . . . .	1280
8.81	ql/FiniteDifferences/bsmtermoperator.hpp File Reference . . . . .	1281
8.82	ql/FiniteDifferences/cranknicolson.hpp File Reference . . . . .	1282
8.83	ql/FiniteDifferences/dminus.hpp File Reference . . . . .	1283
8.84	ql/FiniteDifferences/dplus.hpp File Reference . . . . .	1284
8.85	ql/FiniteDifferences/dplusdminus.hpp File Reference . . . . .	1285
8.86	ql/FiniteDifferences/dzero.hpp File Reference . . . . .	1286
8.87	ql/FiniteDifferences/expliciteuler.hpp File Reference . . . . .	1287
8.88	ql/FiniteDifferences/fdtypedefs.hpp File Reference . . . . .	1288
8.89	ql/FiniteDifferences/finitedifferencemodel.hpp File Reference . . . . .	1289

8.90	<a href="#">ql/FiniteDifferences/impliciteuler.hpp File Reference</a>	1290
8.91	<a href="#">ql/FiniteDifferences/mixedscheme.hpp File Reference</a>	1291
8.92	<a href="#">ql/FiniteDifferences/onefactoroperator.hpp File Reference</a>	1292
8.93	<a href="#">ql/FiniteDifferences/operatorfactory.hpp File Reference</a>	1293
8.94	<a href="#">ql/FiniteDifferences/operatortraits.hpp File Reference</a>	1294
8.95	<a href="#">ql/FiniteDifferences/parallelevolver.hpp File Reference</a>	1295
8.96	<a href="#">ql/FiniteDifferences/pde.hpp File Reference</a>	1296
8.97	<a href="#">ql/FiniteDifferences/pdebsm.hpp File Reference</a>	1297
8.98	<a href="#">ql/FiniteDifferences/pdeshortrate.hpp File Reference</a>	1298
8.99	<a href="#">ql/FiniteDifferences/shoutcondition.hpp File Reference</a>	1299
8.100	<a href="#">ql/FiniteDifferences/stepcondition.hpp File Reference</a>	1300
8.101	<a href="#">ql/FiniteDifferences/tridiagonaloperator.hpp File Reference</a>	1301
8.102	<a href="#">ql/FiniteDifferences/zerocondition.hpp File Reference</a>	1302
8.103	<a href="#">ql/grid.hpp File Reference</a>	1303
8.104	<a href="#">ql/handle.hpp File Reference</a>	1304
8.105	<a href="#">ql/index.hpp File Reference</a>	1305
8.106	<a href="#">ql/Indexes/audlibor.hpp File Reference</a>	1306
8.107	<a href="#">ql/Indexes/cadlibor.hpp File Reference</a>	1307
8.108	<a href="#">ql/Indexes/cdor.hpp File Reference</a>	1308
8.109	<a href="#">ql/Indexes/chflibor.hpp File Reference</a>	1309
8.110	<a href="#">ql/Indexes/dkklibor.hpp File Reference</a>	1310
8.111	<a href="#">ql/Indexes/euribor.hpp File Reference</a>	1311
8.112	<a href="#">ql/Indexes/euriborswapfixa.hpp File Reference</a>	1314
8.113	<a href="#">ql/Indexes/euriborswapfixifr.hpp File Reference</a>	1316
8.114	<a href="#">ql/Indexes/eurlibor.hpp File Reference</a>	1318
8.115	<a href="#">ql/Indexes/eurliborswapfixa.hpp File Reference</a>	1320
8.116	<a href="#">ql/Indexes/eurliborswapfixb.hpp File Reference</a>	1322
8.117	<a href="#">ql/Indexes/eurliborswapfixifr.hpp File Reference</a>	1324
8.118	<a href="#">ql/Indexes/gbplibor.hpp File Reference</a>	1326
8.119	<a href="#">ql/Indexes/indexmanager.hpp File Reference</a>	1327
8.120	<a href="#">ql/Indexes/interestrategyindex.hpp File Reference</a>	1328
8.121	<a href="#">ql/Indexes/jibor.hpp File Reference</a>	1329
8.122	<a href="#">ql/Indexes/jpylibor.hpp File Reference</a>	1330
8.123	<a href="#">ql/Indexes/libor.hpp File Reference</a>	1331
8.124	<a href="#">ql/Indexes/nzdlibor.hpp File Reference</a>	1332
8.125	<a href="#">ql/Indexes/swapindex.hpp File Reference</a>	1333

8.126	ql/Indexes/tibor.hpp File Reference . . . . .	1334
8.127	ql/Indexes/trlibor.hpp File Reference . . . . .	1335
8.128	ql/Indexes/usdlibor.hpp File Reference . . . . .	1336
8.129	ql/Indexes/xibor.hpp File Reference . . . . .	1337
8.130	ql/Indexes/zibor.hpp File Reference . . . . .	1338
8.131	ql/instrument.hpp File Reference . . . . .	1339
8.132	ql/Instruments/asianoption.hpp File Reference . . . . .	1340
8.133	ql/Instruments/assetswap.hpp File Reference . . . . .	1341
8.134	ql/Instruments/barrieroption.hpp File Reference . . . . .	1342
8.135	ql/Instruments/basketoption.hpp File Reference . . . . .	1343
8.136	ql/Instruments/bond.hpp File Reference . . . . .	1344
8.137	ql/Instruments/callabilityschedule.hpp File Reference . . . . .	1345
8.138	ql/Instruments/capfloor.hpp File Reference . . . . .	1346
8.139	ql/Instruments/cliquestoption.hpp File Reference . . . . .	1348
8.140	ql/Instruments/compositeinstrument.hpp File Reference . . . . .	1349
8.141	ql/Instruments/convertiblebond.hpp File Reference . . . . .	1350
8.142	ql/Instruments/dividendschedule.hpp File Reference . . . . .	1352
8.143	ql/Instruments/dividendvanillaoption.hpp File Reference . . . . .	1353
8.144	ql/Instruments/europeanoption.hpp File Reference . . . . .	1354
8.145	ql/Instruments/fixedcouponbond.hpp File Reference . . . . .	1355
8.146	ql/Instruments/fixedcouponbondforward.hpp File Reference . . . . .	1356
8.147	ql/Instruments/floatingratebond.hpp File Reference . . . . .	1357
8.148	ql/Instruments/forward.hpp File Reference . . . . .	1358
8.149	ql/Instruments/forwardrateagreement.hpp File Reference . . . . .	1359
8.150	ql/Instruments/forwardvanillaoption.hpp File Reference . . . . .	1360
8.151	ql/Instruments/lookbackoption.hpp File Reference . . . . .	1361
8.152	ql/Instruments/multiassetoption.hpp File Reference . . . . .	1362
8.153	ql/Instruments/oneassetoption.hpp File Reference . . . . .	1363
8.154	ql/Instruments/oneassetstrikedoption.hpp File Reference . . . . .	1364
8.155	ql/Instruments/payoffs.hpp File Reference . . . . .	1365
8.156	ql/Instruments/quantoforwardvanillaoption.hpp File Reference . . . . .	1366
8.157	ql/Instruments/quantovanillaoption.hpp File Reference . . . . .	1367
8.158	ql/Instruments/stock.hpp File Reference . . . . .	1368
8.159	ql/Instruments/swap.hpp File Reference . . . . .	1369
8.160	ql/Instruments/swaption.hpp File Reference . . . . .	1370
8.161	ql/Instruments/vanillaoption.hpp File Reference . . . . .	1371

8.162	ql/Instruments/vanillaswap.hpp File Reference . . . . .	1372
8.163	ql/Instruments/varianceswap.hpp File Reference . . . . .	1373
8.164	ql/Instruments/zerocouponbond.hpp File Reference . . . . .	1374
8.165	ql/interestrates.hpp File Reference . . . . .	1375
8.166	ql/Lattices/binomialtree.hpp File Reference . . . . .	1376
8.167	ql/Lattices/bsmllattice.hpp File Reference . . . . .	1378
8.168	ql/Lattices/lattice.hpp File Reference . . . . .	1379
8.169	ql/Lattices/lattice1d.hpp File Reference . . . . .	1380
8.170	ql/Lattices/lattice2d.hpp File Reference . . . . .	1381
8.171	ql/Lattices/tflattice.hpp File Reference . . . . .	1382
8.172	ql/Lattices/tree.hpp File Reference . . . . .	1383
8.173	ql/Lattices/trinomialtree.hpp File Reference . . . . .	1384
8.174	ql/MarketModels/driftcalculator.hpp File Reference . . . . .	1385
8.175	ql/Math/array.hpp File Reference . . . . .	1386
8.176	ql/Math/backwardflatinterpolation.hpp File Reference . . . . .	1387
8.177	ql/Math/beta.hpp File Reference . . . . .	1388
8.178	ql/Math/bicubicsplineinterpolation.hpp File Reference . . . . .	1389
8.179	ql/Math/bilinearinterpolation.hpp File Reference . . . . .	1390
8.180	ql/Math/binomialdistribution.hpp File Reference . . . . .	1391
8.181	ql/Math/bivariatenormaldistribution.hpp File Reference . . . . .	1392
8.182	ql/Math/chisquaredistribution.hpp File Reference . . . . .	1393
8.183	ql/Math/choleskydecomposition.hpp File Reference . . . . .	1394
8.184	ql/Math/comparison.hpp File Reference . . . . .	1395
8.185	ql/Math/convergencestatistics.hpp File Reference . . . . .	1396
8.186	ql/Math/cubicspline.hpp File Reference . . . . .	1397
8.187	ql/Math/discrepancystatistics.hpp File Reference . . . . .	1398
8.188	ql/Math/errorfunction.hpp File Reference . . . . .	1399
8.189	ql/Math/extrapolation.hpp File Reference . . . . .	1400
8.190	ql/Math/factorial.hpp File Reference . . . . .	1401
8.191	ql/Math/forwardflatinterpolation.hpp File Reference . . . . .	1402
8.192	ql/Math/functional.hpp File Reference . . . . .	1403
8.193	ql/Math/gammadistribution.hpp File Reference . . . . .	1404
8.194	ql/Math/gaussianorthogonalpolynomial.hpp File Reference . . . . .	1405
8.195	ql/Math/gaussianquadratures.hpp File Reference . . . . .	1406
8.196	ql/Math/gaussianstatistics.hpp File Reference . . . . .	1408
8.197	ql/Math/generalstatistics.hpp File Reference . . . . .	1409

8.198	ql/Math/incompletegamma.hpp File Reference . . . . .	1410
8.199	ql/Math/incrementalstatistics.hpp File Reference . . . . .	1411
8.200	ql/Math/interpolation.hpp File Reference . . . . .	1412
8.201	ql/Math/interpolation2D.hpp File Reference . . . . .	1413
8.202	ql/Math/kronrodintegral.hpp File Reference . . . . .	1414
8.203	ql/Math/lexicographicalview.hpp File Reference . . . . .	1415
8.204	ql/Math/linearinterpolation.hpp File Reference . . . . .	1416
8.205	ql/Math/linearleastquaresregression.hpp File Reference . . . . .	1417
8.206	ql/Math/loglinearinterpolation.hpp File Reference . . . . .	1418
8.207	ql/Math/matrix.hpp File Reference . . . . .	1419
8.208	ql/Math/multicubicspline.hpp File Reference . . . . .	1420
8.209	ql/Math/normaldistribution.hpp File Reference . . . . .	1422
8.210	ql/Math/poissondistribution.hpp File Reference . . . . .	1423
8.211	ql/Math/primenumbers.hpp File Reference . . . . .	1424
8.212	ql/Math/pseudosqrt.hpp File Reference . . . . .	1425
8.213	ql/Math/riskstatistics.hpp File Reference . . . . .	1426
8.214	ql/Math/rounding.hpp File Reference . . . . .	1427
8.215	ql/Math/sabrinterpolation.hpp File Reference . . . . .	1428
8.216	ql/Math/sampledcurve.hpp File Reference . . . . .	1429
8.217	ql/Math/segmentintegral.hpp File Reference . . . . .	1430
8.218	ql/Math/sequencestatistics.hpp File Reference . . . . .	1431
8.219	ql/Math/simpsonintegral.hpp File Reference . . . . .	1433
8.220	ql/Math/statistics.hpp File Reference . . . . .	1434
8.221	ql/Math/svd.hpp File Reference . . . . .	1435
8.222	ql/Math/symmetricschurdecomposition.hpp File Reference . . . . .	1436
8.223	ql/Math/tqreigendecomposition.hpp File Reference . . . . .	1437
8.224	ql/Math/transformedgrid.hpp File Reference . . . . .	1438
8.225	ql/Math/trapezoidintegral.hpp File Reference . . . . .	1439
8.226	ql/money.hpp File Reference . . . . .	1440
8.227	ql/MonteCarlo/brownianbridge.hpp File Reference . . . . .	1441
8.228	ql/MonteCarlo/earlyexercisepathpricer.hpp File Reference . . . . .	1442
8.229	ql/MonteCarlo/getcovariance.hpp File Reference . . . . .	1443
8.230	ql/MonteCarlo/longstaffschwartzpathpricer.hpp File Reference . . . . .	1444
8.231	ql/MonteCarlo/lsmbasissystem.hpp File Reference . . . . .	1445
8.232	ql/MonteCarlo/mctraits.hpp File Reference . . . . .	1446
8.233	ql/MonteCarlo/mctypedefs.hpp File Reference . . . . .	1447

8.234	ql/MonteCarlo/montecarlomodel.hpp File Reference . . . . .	1448
8.235	ql/MonteCarlo/multipath.hpp File Reference . . . . .	1449
8.236	ql/MonteCarlo/multipathgenerator.hpp File Reference . . . . .	1450
8.237	ql/MonteCarlo/path.hpp File Reference . . . . .	1451
8.238	ql/MonteCarlo/pathgenerator.hpp File Reference . . . . .	1452
8.239	ql/MonteCarlo/pathpricer.hpp File Reference . . . . .	1453
8.240	ql/MonteCarlo/sample.hpp File Reference . . . . .	1454
8.241	ql/numericalmethod.hpp File Reference . . . . .	1455
8.242	ql/Optimization/armijo.hpp File Reference . . . . .	1456
8.243	ql/Optimization/conjugategradient.hpp File Reference . . . . .	1457
8.244	ql/Optimization/constraint.hpp File Reference . . . . .	1458
8.245	ql/Optimization/costfunction.hpp File Reference . . . . .	1459
8.246	ql/Optimization/criteria.hpp File Reference . . . . .	1460
8.247	ql/Optimization/leastsquare.hpp File Reference . . . . .	1461
8.248	ql/Optimization/levenbergmarquardt.hpp File Reference . . . . .	1462
8.249	ql/Optimization/linesearch.hpp File Reference . . . . .	1463
8.250	ql/Optimization/lmdif.hpp File Reference . . . . .	1464
8.251	ql/Optimization/method.hpp File Reference . . . . .	1465
8.252	ql/Optimization/problem.hpp File Reference . . . . .	1466
8.253	ql/Optimization/simplex.hpp File Reference . . . . .	1467
8.254	ql/Optimization/steepestdescent.hpp File Reference . . . . .	1468
8.255	ql/option.hpp File Reference . . . . .	1469
8.256	ql/Patterns/bridge.hpp File Reference . . . . .	1470
8.257	ql/Patterns/composite.hpp File Reference . . . . .	1471
8.258	ql/Patterns/curiouslyrecurring.hpp File Reference . . . . .	1472
8.259	ql/Patterns/lazyobject.hpp File Reference . . . . .	1473
8.260	ql/Patterns/observable.hpp File Reference . . . . .	1474
8.261	ql/Patterns/singleton.hpp File Reference . . . . .	1475
8.262	ql/Patterns/visitor.hpp File Reference . . . . .	1476
8.263	ql/payoff.hpp File Reference . . . . .	1477
8.264	ql/period.hpp File Reference . . . . .	1478
8.265	ql/position.hpp File Reference . . . . .	1480
8.266	ql/Pricers/discretegeometriccaso.hpp File Reference . . . . .	1481
8.267	ql/Pricers/mccliquetoption.hpp File Reference . . . . .	1482
8.268	ql/Pricers/mcdiscretearithmeticcaso.hpp File Reference . . . . .	1483
8.269	ql/Pricers/mceverest.hpp File Reference . . . . .	1484

8.270	ql/Pricers/mchimalaya.hpp File Reference . . . . .	1485
8.271	ql/Pricers/mcmaxbasket.hpp File Reference . . . . .	1486
8.272	ql/Pricers/mcpagoda.hpp File Reference . . . . .	1487
8.273	ql/Pricers/mcperformanceoption.hpp File Reference . . . . .	1488
8.274	ql/Pricers/mcpricer.hpp File Reference . . . . .	1489
8.275	ql/Pricers/singleassetoption.hpp File Reference . . . . .	1490
8.276	ql/prices.hpp File Reference . . . . .	1491
8.277	ql/pricingengine.hpp File Reference . . . . .	1492
8.278	ql/PricingEngines/americanpayoffatexpiry.hpp File Reference . . . . .	1493
8.279	ql/PricingEngines/americanpayoffathit.hpp File Reference . . . . .	1494
8.280	ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp File Reference . . . .	1495
8.281	ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp File Reference . . .	1496
8.282	ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp File Reference . . . . .	1497
8.283	ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp File Reference . . . . .	1498
8.284	ql/PricingEngines/Asian/mcdiscreteasianengine.hpp File Reference . . . . .	1499
8.285	ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference . . . . .	1500
8.286	ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference . . . . .	1501
8.287	ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference . . . . .	1502
8.288	ql/PricingEngines/Basket/mcbasketengine.hpp File Reference . . . . .	1503
8.289	ql/PricingEngines/Basket/stulzengine.hpp File Reference . . . . .	1504
8.290	ql/PricingEngines/blackformula.hpp File Reference . . . . .	1505
8.291	ql/PricingEngines/blackmodel.hpp File Reference . . . . .	1506
8.292	ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference . . . . .	1507
8.293	ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference . . . . .	1508
8.294	ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference . . . . .	1509
8.295	ql/PricingEngines/CapFloor/mchullwhiteengine.hpp File Reference . . . . .	1510
8.296	ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference . . . . .	1511
8.297	ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference . . . . .	1512
8.298	ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference . . . .	1513
8.299	ql/PricingEngines/Forward/forwardengine.hpp File Reference . . . . .	1514
8.300	ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference . . .	1515
8.301	ql/PricingEngines/Forward/mcvarianceswapengine.hpp File Reference . . . . .	1516
8.302	ql/PricingEngines/Forward/replicatingvarianceswapengine.hpp File Reference .	1517
8.303	ql/PricingEngines/genericmodelengine.hpp File Reference . . . . .	1518
8.304	ql/PricingEngines/greeks.hpp File Reference . . . . .	1519
8.305	ql/PricingEngines/Hybrid/binomialconvertibleengine.hpp File Reference . . . .	1520



8.306	ql/PricingEngines/Hybrid/discretizedconvertible.hpp File Reference . . . . .	1521
8.307	ql/PricingEngines/latticeshortratemodelengine.hpp File Reference . . . . .	1522
8.308	ql/PricingEngines/Lookback/analyticcontinuousfixedlookback.hpp File Reference	1523
8.309	ql/PricingEngines/Lookback/analyticcontinuousfloatinglookback.hpp File Reference . . . . .	1524
8.310	ql/PricingEngines/mclongstaffschwartzengine.hpp File Reference . . . . .	1525
8.311	ql/PricingEngines/mcsimulation.hpp File Reference . . . . .	1526
8.312	ql/PricingEngines/Quanto/quantoengine.hpp File Reference . . . . .	1527
8.313	ql/PricingEngines/Swaption/blackswaptionengine.hpp File Reference . . . . .	1528
8.314	ql/PricingEngines/Swaption/discretizedswaption.hpp File Reference . . . . .	1529
8.315	ql/PricingEngines/Swaption/g2swaptionengine.hpp File Reference . . . . .	1530
8.316	ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp File Reference . . .	1531
8.317	ql/PricingEngines/Swaption/lfmswaptionengine.hpp File Reference . . . . .	1532
8.318	ql/PricingEngines/Swaption/treeswaptionengine.hpp File Reference . . . . .	1533
8.319	ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference . .	1534
8.320	ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp File Reference .	1535
8.321	ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference . . . . .	1536
8.322	ql/PricingEngines/Vanilla/analytichestonengine.hpp File Reference . . . . .	1537
8.323	ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference . . . . .	1538
8.324	ql/PricingEngines/Vanilla/batesengine.hpp File Reference . . . . .	1539
8.325	ql/PricingEngines/Vanilla/binomialengine.hpp File Reference . . . . .	1540
8.326	ql/PricingEngines/Vanilla/bjerkhundstenglandengine.hpp File Reference . . . . .	1541
8.327	ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference . . . . .	1542
8.328	ql/PricingEngines/Vanilla/fdamericanengine.hpp File Reference . . . . .	1543
8.329	ql/PricingEngines/Vanilla/fdbermudanengine.hpp File Reference . . . . .	1544
8.330	ql/PricingEngines/Vanilla/fdconditions.hpp File Reference . . . . .	1545
8.331	ql/PricingEngines/Vanilla/fddividendamericanengine.hpp File Reference . . . .	1546
8.332	ql/PricingEngines/Vanilla/fddividendengine.hpp File Reference . . . . .	1547
8.333	ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp File Reference . . . .	1548
8.334	ql/PricingEngines/Vanilla/fddividendshoutengine.hpp File Reference . . . . .	1549
8.335	ql/PricingEngines/Vanilla/fdeuropeanengine.hpp File Reference . . . . .	1550
8.336	ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp File Reference . . . . .	1551
8.337	ql/PricingEngines/Vanilla/fdshoutengine.hpp File Reference . . . . .	1552
8.338	ql/PricingEngines/Vanilla/fdstepconditionengine.hpp File Reference . . . . .	1553
8.339	ql/PricingEngines/Vanilla/fdvanillaengine.hpp File Reference . . . . .	1554
8.340	ql/PricingEngines/Vanilla/integralengine.hpp File Reference . . . . .	1555
8.341	ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference . . . . .	1556



8.342	ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference . . . . .	1557
8.343	ql/PricingEngines/Vanilla/mcamericanengine.hpp File Reference . . . . .	1558
8.344	ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference . . . . .	1559
8.345	ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference . . . . .	1560
8.346	ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp File Reference . . . . .	1561
8.347	ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference . . . . .	1562
8.348	ql/Processes/blackscholesprocess.hpp File Reference . . . . .	1563
8.349	ql/Processes/eulerdiscretization.hpp File Reference . . . . .	1564
8.350	ql/Processes/forwardmeasureprocess.hpp File Reference . . . . .	1565
8.351	ql/Processes/g2process.hpp File Reference . . . . .	1566
8.352	ql/Processes/geometricbrownianprocess.hpp File Reference . . . . .	1567
8.353	ql/Processes/hestonprocess.hpp File Reference . . . . .	1568
8.354	ql/Processes/hullwhiteprocess.hpp File Reference . . . . .	1569
8.355	ql/Processes/lfmcovarParams.hpp File Reference . . . . .	1570
8.356	ql/Processes/lfmhullwhiteparam.hpp File Reference . . . . .	1571
8.357	ql/Processes/lfmprocess.hpp File Reference . . . . .	1572
8.358	ql/Processes/merton76process.hpp File Reference . . . . .	1573
8.359	ql/Processes/ornsteinuhlenbeckprocess.hpp File Reference . . . . .	1574
8.360	ql/Processes/squarerootprocess.hpp File Reference . . . . .	1575
8.361	ql/Processes/stochasticprocessarray.hpp File Reference . . . . .	1576
8.362	ql/qldefines.hpp File Reference . . . . .	1577
8.363	ql/quote.hpp File Reference . . . . .	1579
8.364	ql/RandomNumbers/boxmullergaussianrng.hpp File Reference . . . . .	1580
8.365	ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference . . . . .	1581
8.366	ql/RandomNumbers/faurersg.hpp File Reference . . . . .	1582
8.367	ql/RandomNumbers/haltonrng.hpp File Reference . . . . .	1583
8.368	ql/RandomNumbers/inversecumulativerng.hpp File Reference . . . . .	1584
8.369	ql/RandomNumbers/inversecumulativersg.hpp File Reference . . . . .	1585
8.370	ql/RandomNumbers/knuthuniformrng.hpp File Reference . . . . .	1586
8.371	ql/RandomNumbers/lecuyeruniformrng.hpp File Reference . . . . .	1587
8.372	ql/RandomNumbers/mt19937uniformrng.hpp File Reference . . . . .	1588
8.373	ql/RandomNumbers/randomizedlds.hpp File Reference . . . . .	1589
8.374	ql/RandomNumbers/randomsequencegenerator.hpp File Reference . . . . .	1590
8.375	ql/RandomNumbers/rngtraits.hpp File Reference . . . . .	1591
8.376	ql/RandomNumbers/seedgenerator.hpp File Reference . . . . .	1593
8.377	ql/RandomNumbers/sobolrng.hpp File Reference . . . . .	1594

8.378	ql/schedule.hpp File Reference . . . . .	1595
8.379	ql/settings.hpp File Reference . . . . .	1596
8.380	ql/ShortRateModels/calibrationhelper.hpp File Reference . . . . .	1597
8.381	ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference . . . . .	1598
8.382	ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp File Reference .	1599
8.383	ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference . . .	1600
8.384	ql/ShortRateModels/LiborMarketModels/lfmcovarproxy.hpp File Reference . . .	1601
8.385	ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp File Reference	1602
8.386	ql/ShortRateModels/LiborMarketModels/lmconstwrappercorrmodel.hpp File Reference . . . . .	1603
8.387	ql/ShortRateModels/LiborMarketModels/lmconstwrappervolmodel.hpp File Reference . . . . .	1604
8.388	ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp File Reference . . . .	1605
8.389	ql/ShortRateModels/LiborMarketModels/lmexpcorrmodel.hpp File Reference . .	1606
8.390	ql/ShortRateModels/LiborMarketModels/lmextlinexpvolmodel.hpp File Reference	1607
8.391	ql/ShortRateModels/LiborMarketModels/lmfixedvolmodel.hpp File Reference .	1608
8.392	ql/ShortRateModels/LiborMarketModels/lmlinexpcorrmodel.hpp File Reference	1609
8.393	ql/ShortRateModels/LiborMarketModels/lmlinexpvolmodel.hpp File Reference	1610
8.394	ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp File Reference . . . .	1611
8.395	ql/ShortRateModels/model.hpp File Reference . . . . .	1612
8.396	ql/ShortRateModels/onefactormodel.hpp File Reference . . . . .	1613
8.397	ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference . . .	1614
8.398	ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference . . .	1615
8.399	ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Ref- erence . . . . .	1616
8.400	ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference . . . . .	1617
8.401	ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference . . . . .	1618
8.402	ql/ShortRateModels/parameter.hpp File Reference . . . . .	1619
8.403	ql/ShortRateModels/twofactormodel.hpp File Reference . . . . .	1620
8.404	ql/ShortRateModels/TwoFactorModels/batesmodel.hpp File Reference . . . . .	1621
8.405	ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference . . . . .	1622
8.406	ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp File Reference . . . . .	1623
8.407	ql/solver1d.hpp File Reference . . . . .	1624
8.408	ql/Solvers1D/bisection.hpp File Reference . . . . .	1625
8.409	ql/Solvers1D/brent.hpp File Reference . . . . .	1626
8.410	ql/Solvers1D/falseposition.hpp File Reference . . . . .	1627
8.411	ql/Solvers1D/newton.hpp File Reference . . . . .	1628

8.412	ql/Solvers1D/newtonsafe.hpp File Reference . . . . .	1629
8.413	ql/Solvers1D/ridder.hpp File Reference . . . . .	1630
8.414	ql/Solvers1D/secant.hpp File Reference . . . . .	1631
8.415	ql/stochasticprocess.hpp File Reference . . . . .	1632
8.416	ql/swaptionvolstructure.hpp File Reference . . . . .	1633
8.417	ql/termstructure.hpp File Reference . . . . .	1634
8.418	ql/TermStructures/bondhelpers.hpp File Reference . . . . .	1635
8.419	ql/TermStructures/bootstraptraits.hpp File Reference . . . . .	1636
8.420	ql/TermStructures/compoundforward.hpp File Reference . . . . .	1637
8.421	ql/TermStructures/discountcurve.hpp File Reference . . . . .	1638
8.422	ql/TermStructures/drifttermstructure.hpp File Reference . . . . .	1639
8.423	ql/TermStructures/extendeddiscountcurve.hpp File Reference . . . . .	1640
8.424	ql/TermStructures/flatforward.hpp File Reference . . . . .	1641
8.425	ql/TermStructures/forwardcurve.hpp File Reference . . . . .	1642
8.426	ql/TermStructures/forwardspreadedtermstructure.hpp File Reference . . . . .	1643
8.427	ql/TermStructures/forwardstructure.hpp File Reference . . . . .	1644
8.428	ql/TermStructures/impliedtermstructure.hpp File Reference . . . . .	1645
8.429	ql/TermStructures/piecewiseflatforward.hpp File Reference . . . . .	1646
8.430	ql/TermStructures/piecewiseyieldcurve.hpp File Reference . . . . .	1647
8.431	ql/TermStructures/piecewisezerospreadedtermstructure.hpp File Reference . . . . .	1648
8.432	ql/TermStructures/quantotermstructure.hpp File Reference . . . . .	1649
8.433	ql/TermStructures/ratehelpers.hpp File Reference . . . . .	1650
8.434	ql/TermStructures/zerocurve.hpp File Reference . . . . .	1651
8.435	ql/TermStructures/zerospreadedtermstructure.hpp File Reference . . . . .	1652
8.436	ql/TermStructures/zeroyieldstructure.hpp File Reference . . . . .	1653
8.437	ql/timegrid.hpp File Reference . . . . .	1654
8.438	ql/timeseries.hpp File Reference . . . . .	1655
8.439	ql/types.hpp File Reference . . . . .	1656
8.440	ql/Utilities/clone.hpp File Reference . . . . .	1658
8.441	ql/Utilities/dataformatters.hpp File Reference . . . . .	1659
8.442	ql/Utilities/dataparsers.hpp File Reference . . . . .	1660
8.443	ql/Utilities/disposable.hpp File Reference . . . . .	1661
8.444	ql/Utilities/null.hpp File Reference . . . . .	1662
8.445	ql/Utilities/observablevalue.hpp File Reference . . . . .	1663
8.446	ql/Utilities/steppingiterator.hpp File Reference . . . . .	1664
8.447	ql/Utilities/strings.hpp File Reference . . . . .	1665

8.448	ql/Utilities/tracing.hpp File Reference . . . . .	1666
8.449	ql/Volatilities/blackconstantvol.hpp File Reference . . . . .	1668
8.450	ql/Volatilities/blackvariancecurve.hpp File Reference . . . . .	1669
8.451	ql/Volatilities/blackvariancesurface.hpp File Reference . . . . .	1670
8.452	ql/Volatilities/capflatvolvector.hpp File Reference . . . . .	1671
8.453	ql/Volatilities/capletconstantvol.hpp File Reference . . . . .	1672
8.454	ql/Volatilities/capletvariancecurve.hpp File Reference . . . . .	1673
8.455	ql/Volatilities/cmsmarket.hpp File Reference . . . . .	1674
8.456	ql/Volatilities/impliedvoltermstructure.hpp File Reference . . . . .	1675
8.457	ql/Volatilities/localconstantvol.hpp File Reference . . . . .	1676
8.458	ql/Volatilities/localvolcurve.hpp File Reference . . . . .	1677
8.459	ql/Volatilities/localvolsurface.hpp File Reference . . . . .	1678
8.460	ql/Volatilities/smilesection.hpp File Reference . . . . .	1679
8.461	ql/Volatilities/swaptionconstantvol.hpp File Reference . . . . .	1680
8.462	ql/Volatilities/swaptionvolcube.hpp File Reference . . . . .	1681
8.463	ql/Volatilities/swaptionvolcubebysabr.hpp File Reference . . . . .	1682
8.464	ql/Volatilities/swaptionvolmatrix.hpp File Reference . . . . .	1683
8.465	ql/volatilitymodel.hpp File Reference . . . . .	1684
8.466	ql/VolatilityModels/constantestimator.hpp File Reference . . . . .	1685
8.467	ql/VolatilityModels/garch.hpp File Reference . . . . .	1686
8.468	ql/VolatilityModels/garmanklass.hpp File Reference . . . . .	1687
8.469	ql/VolatilityModels/simplelocalestimator.hpp File Reference . . . . .	1688
8.470	ql/voltermstructure.hpp File Reference . . . . .	1689
8.471	ql/yieldtermstructure.hpp File Reference . . . . .	1690
<b>9</b>	<b>QuantLib Example Documentation</b>	<b>1691</b>
9.1	BermudanSwaption.cpp . . . . .	1691
9.2	ConvertibleBonds.cpp . . . . .	1697
9.3	DiscreteHedging.cpp . . . . .	1702
9.4	EquityOption.cpp . . . . .	1708
9.5	FRA.cpp . . . . .	1714
9.6	Replication.cpp . . . . .	1720
9.7	Repo.cpp . . . . .	1726
9.8	swapvaluation.cpp . . . . .	1730
9.9	tracing_example.cpp . . . . .	1742
<b>10</b>	<b>Caveats</b>	<b>1745</b>

<b>11 Test Suite</b>	<b><a href="#">1753</a></b>
<b>12 Todo List</b>	<b><a href="#">1765</a></b>
<b>13 Known Bugs</b>	<b><a href="#">1769</a></b>
<b>14 Deprecated List</b>	<b><a href="#">1771</a></b>



# Chapter 1

## Getting started

### 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 License](#).

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 [QuantLib License](#) for more details.

#### 1.1.1 Disclaimer

At this time, this documentation is widely incomplete and must be regarded as a work in progress. Contributions are welcome.

## 1.2 Project overview

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

The following list is a (possibly outdated) overview of the existing code base.

The [QuantLib-users](#) and [QuantLib-dev](#) mailing lists are the preferred forum for proposals, suggestions and contributions regarding the future development of the library.

### Date, calendars, and day count conventions

- Date class.
- Weekday, month, frequency, time unit enumerations.
- Period class (eg. 1y, 30d, 2m, etc.)
- IMM calculation.
- More than 30 business calendars.
- NullCalendar (no holidays) for theoretical calculations.
- Joint calendars made up as holiday union or intersection of base calendars.
- Rolling conventions: Preceding, ModifiedPreceding, Following, ModifiedFollowing, MonthEndReference.
- Schedule class for date stream generation.
- Day count conventions: Actual360, Actual365Fixed, ActualActual (Bond, ISDA, AFB), 30/360 (US, European, Italian), 1/1.

To do:

- Differentiate more calendars depending on country or exchange, instead of city.
- enable business day calculation in addition to calendar days calculation in `DayCounter::daycount()`. See `DayTypeEnum` in `FpML`.

### Math

- Linear, log-linear, and cubic spline interpolation.
- Primitive, first and second derivative functions of cubic and linear interpolators.
- Cubic spline end conditions: first derivative value, second derivative value, not-a-knot.
- Monotone cubic spline with Hyman non-restrictive filter.
- Bicubic spline and bilinear interpolations.
- N-dimensional cubic spline interpolation.
- Normal and cumulative normal distributions.
- Inverse cumulative normal distribution: Moro and Acklam approximations.
- Bivariate cumulative normal distribution.
- Binomial coefficients, binomial distribution, cumulative binomial distribution, and Peizer-Pratt inversion (method 2.)



- Chi square and non-central chi square distributions.
- Beta functions.
- Poisson and cumulative Poisson distributions.
- Incomplete gamma functions.
- Gamma distribution.
- Factorials.
- Integration algorithms: segment, trapezoid, mid-point trapezoid, Simpson, Gauss-Kronrod.
- Error function.
- General 1-D statistics: mean, variance, standard deviation, skewness, kurtosis, error estimation, min, max.
- Multi-dimensional (sequence) statistics: all the 1-D methods plus covariance, correlation, L2-discrepancy calculation, etc.
- Risk measures for Gaussian and empirical distributions: semi-variance, regret, percentile, top percentile, value-at-risk, upside potential, shortfall, average shortfall, expected shortfall.
- Array and matrix classes for algebra.
- Singular value decomposition.
- Eigenvalues, eigenvectors for symmetric matrices.
- Cholesky decomposition.
- Schur decomposition.
- Spectral rank-reduced square root, spectral pseudo-square root.

To do:

- Periodic and Lagrange end conditions for cubic spline.
- Implement convexity-preserving filter for cubic spline.
- Log-linear interpolator primitive, first and second derivative functions.
- Revise end conditions for bicubic and N-dimensional spline.
- Trivariate and multi-variate distribution, see Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.
- Hypersphere decomposition, Higham algorithm for pseudo-square root.
- interface with GALib (genetic algorithm)
- Add COOOL algorithms
- Histogram class

### 1-dimensional solvers

- Bisection, false position, Newton, bounded Newton, Ridder, secant, Brent.

To do:

- Clean up the interface so that it is clear whether the accuracy is specified for  $x$  or  $f(x)$ .

### Optimization

- Conjugate gradient, simplex, steepest descent, line search, Armijo line search, least squares.
- Constrained (positive, boundary, etc.) and unconstrained optimization

### Random-number generation

- Uniform pseudo-random sequences: Knuth, L'Ecuyer, Mersenne twister.
- Uniform quasi-random (low-discrepancy) sequences: Halton, Faure, Sobol up to dimension 21,200 (8,129,334 if you really want) with unit, Jäkel, Bradley-Fox, and Lemieux-Cieslak-Luttmer initialization numbers.
- Randomized quasi-random sequences (in progress)
- Randomized (shifted) low-discrepancy sequences.
- Primitive polynomials modulo 2 up to dimension 18 (available up to dimension 27)
- Gaussian random numbers from uniform random numbers using different algorithms: central limit theorem, Box-Muller, inverse cumulative (Moro and Acklam algorithms)

### Patterns

- Bridge, composite, lazy object, observer/observable, singleton, strategy, visitor.

### Finite differences

- Mixed theta, implicit, explicit, and Crank-Nicolson 1-dimensional schemes.
- Differential operators:  $D_0$ ,  $D_+$ ,  $D_-$ ,  $D_+D_-$ .
- Shout, Bermudan and American exercises.

To do:

- Richardson extrapolation
- Introduce variable theta schemes.
- Introduce multi time-level schemes.
- Enable different solvers (SOR, etc.)
- Extend to time-dependant parameters.
- Extend to local volatility.
- Two-dimension schemes.
- Improve boundary conditions.
- Use DiscretisedAsset instead of array?

- Use TimeGrid
- Use assetGrid
- Handle barrier specification
- Handle variable asset step size
- Check (and improve) vega, rho, dividendRho greeks, solving their own equations

### Lattices

- Binomial trees: Cox-Ross-Rubinstein, Jarrow-Rudd, additive equiprobabilities, Trigeorgis, Tian, Leisen-Reimer.
- Trinomial (interest-rate) tree.
- Discretized asset.
- Richardson extrapolation

To do:

- Merge finite differences with the lattice framework. Use same rollback scheme.
- Trinomial trees
- Implied trinomial trees
- Calculate binomial tree greeks

### Monte Carlo

- One-factor and multi-factor path classes.
- Path-generator classes: incremental and Brownian-bridge one-factor path generation, incremental multi-factor path generation.
- General-purpose Monte Carlo model based on traits for path samples.
- Antithetic variance-reduction technique.
- Control variate technique.

To do:

- Greeks calculation.
- Allow easier selection between Incremental/BrownianBridge path generation.
- Review Monte Carlo engine for american options.
- Review multi-factor Monte Carlo simulation.
- Predictor-corrector scheme.
- Add Milstein scheme.
- Add Martingale control variate.

- Batch samples as  $N=n\_batches*batch\_size$ , and exploit it for randomized low discrepancy sequences (RQMC)

### Pricing engines

- Analytic Black formula (plus greeks) for different payoffs.
- Analytic formula for American-style digital options with payoff at expiry.
- Analytic formula for American-style digital options with payoff at hit.
- Monte Carlo simulation base engine.
- Lattice short rate model base engine.
- Engines for options described by "vanilla" set of parameters: analytic digital American, analytic discrete-dividend European, analytic European, Barone-Adesi and Whaley approximation for American, Ju approximation for American, binomial (Cox-Ross-Rubinstein, Jarrow-Rudd, additive equiprobabilities, Trigeorgis, Tian, Leisen-Reimer), Bjerk Sund and Stensland approximation for American, integral European, Merton 76 jump-diffusion, Monte Carlo digital, Monte Carlo European, Bates and Heston models, finite-difference European, Bermudan and American.
- Engines for options described by "barrier" set of parameters: analytic down/up in/out, Monte Carlo down/up in/out
- Engines for Asian options: analytic discrete geometric average-price, analytic continuous geometric average-price, Monte Carlo discrete arithmetic average-price, Monte Carlo discrete geometric average-price.
- Engines for options described by "cliquet" set of parameters: analytic, analytic performance.
- Forward and forward-performance compound engines.
- Quanto compound engine.
- Quanto-forward and Quanto-forward-performance compound engines.
- Basket engine: analytic Stulz engine for max/min on two assets, Monte Carlo engine (in progress).
- Black model base class for vanilla interest rate derivatives
- Cap/floor pricing engines: analytic Black model, analytic affine models, tree based engine.
- Swaption pricing engines: analytic Black model, analytic affine models (Jamshidian), tree based engine.

To do:

- Add the trigger level Touch/NoTouch specification for American-style digitals.
- More vanilla engines: Roll-Geske-Whaley American Call, Geske-Johnson American Put, finite differences, Edgeworth expansion binomial tree, etc.
- Merge NesQuant SJD engine (<http://www.nielses.dk/quantlib/nesquant/>)
- Continuous geometric average-strike.
- Ensure all path-dependent options allow for evaluation with collected past observations.

- Define dividendRho for discrete dividends.

### Pricers

- Cliquet option
- Analytic discrete geometric average-price option (European exercise).
- Analytic discrete geometric average-strike option (European exercise).
- Monte Carlo cliquet option.
- Monte Carlo discrete arithmetic average-price option.
- Monte Carlo discrete arithmetic average-strike option.
- Monte Carlo Everest option.
- Monte Carlo Himalaya option.
- Monte Carlo max basket option.
- Monte Carlo pagoda option.
- Monte Carlo forward performance option.

### Financial Instruments

- Instrument base class: npv(), isExpired(), etc.
- Interest-rate swap: simple, normal.
- Swaption.
- Cap/floor.
- Fixed-rate coupon bond.
- Stock.
- One-asset option base class.
- Asian option.
- Barrier option.
- Cliquet option.
- Forward vanilla option.
- Quanto vanilla option.
- Quanto-forward vanilla option.
- Vanilla option.
- Multi-asset option base class.
- Basket option.

To do:

- Forward (stock) and FRA (forward-rate agreement).
- More bonds.

#### **Yield term structures**

- Term structure common interface.
- Term structure classes based on discount, zero, or forward underlying description.
- Term structure based on linear interpolation of zero yields.
- Term structure based on log-linear interpolation of discounts.
- Term structure based on constant flat forward.
- Term structure based on piecewise-constant flat forwards with libor-futures-swap bootstrapping algorithm.
- Spreaded term structures.
- Forward-date implied term structure.

To do:

- Future convexity adjustment
- End of year effect

#### **Volatility**

- Interface for cap/floor Black volatility term structures (unstable).
- Interface for swaption Black volatility term structures (unstable).
- Interface for equity Black volatility term structures based on volatility or variance underlying description: constant, time-dependant curve, time-strike surface, forward date implied term structure.
- Interface for equity local volatility term structures: constant, time-dependant curve, time-asset level surface (Gatheral's formula).

To do:

- Fix implementation of Gatheral's formula for local volatility.

#### **Short rate models**

- Single factor models: Hull-White, Black-Karasinski, Vasicek (untested), CIR (untested), Extended CIR (untested).
- Two factor models: G2 (untested).

#### **Credit derivatives**

To do:

- everything.

**Test suite**

Implemented by means of the Boost unit-test framework. More than 210 automated tests. An automatically-generated list is available in chapter [11](#).

To do:

- Add covariance/correlation test for SequenceStatistics.
- Increase coverage.

**Miscellanea**

- Index classes for handling of fixed-income libor indexes (fixings, forecasting, etc.)
- Cash-flow class.
- Currency class and enumeration.
- Money class with automatic exchange-rate capabilities.
- Output data formatters: long integers, Ordinal numerals, power of two, exponential, fixed digit, sequences, dates, etc.
- Input data parsers.
- Error classes and error handling.
- Exercise classes: European, Bermudan, American
- Payoff classes: plain, gap, asset-or-nothing, cash-or-nothing
- Grid classes for handling of equally and unequally spaced grids.
- History class for handling of historical data.
- Quote class for mutable data.
- Null types.
- User-configurable flag to disable usage of deprecated classes.

To do:

- Implement currency as per OMG definition

**Documentation**

- Documentation automatically generated with Doxygen (html, PDF, ps, WinHelp, man pages)

To do:

- Add a "Getting started" page to the site

## 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.shtml>

### 1.3.2 Current CVS snapshot

Instructions for anonymous CVS access are available at <http://quantlib.org/cvs.shtml>

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

Before installing QuantLib, make sure that you have a working Boost installation; see [http://www.boost.org/more/getting\\_started.html](http://www.boost.org/more/getting_started.html) for instructions. Boost 1.31 or later is required; Boost 1.33 is suggested.

### 1.4.1 Linux/Unix/Mac OS X/Cygwin

A tarball of the source distribution is available from

<http://quantlib.org/download.shtml>

After uncompressing the sources:

1. 'cd' to the QuantLib directory and type './configure' to configure the package for your system; see the [User configuration](#) section for configuration options.
2. Type 'make' to compile the package.
3. Type 'make install' to install the library. This might require administrative privileges.

### 1.4.2 Win32

An installer for the source distribution is available at

<http://quantlib.org/download.shtml>

Before compiling the library, you might want to edit the file "ql/userconfig.hpp"; see the [User configuration](#) section for details.

Visual C++ 6.0/7.1/8.0 projects files are supplied for building the library.

Dev-C++ project files are provided in order to make easier the usage of Mingw/GCC under Win32.

If you use Borland command line compiler (5.5.1) the make options are: -D\_DEBUG (debug), -D\_RTLDLL (dynamic linking of runtime library), -D\_MT\_\_ (multi-thread) as in:

```
make all
```

```
make -D_DEBUG all
```

```
make -D__MT__ -D_RTLDLL -D_DEBUG all
```

or any other combination of options.

## 1.5 User configuration

A number of macros is provided for user configuration. Defining or undefining such macros triggers variations in some library functionality.

Under a Linux/Unix system, they are (un)set by `configure`; run

```
./configure --help
```

for a list of corresponding command-line options.

Under a Windows system, they must be (un)defined by editing the file `<ql/userconfig.hpp>` and commenting or uncommenting the relevant lines.

Such macros include:

```
#define QL_ERROR_FUNCTIONS
```

If defined, function information is added to the error messages thrown by the library. Undefined by default.

```
#define QL_ERROR_LINES
```

If defined, file and line information is added to the error messages thrown by the library. Undefined by default.

```
#define QL_ENABLE_TRACING
```

If enabled, tracing messages might be emitted by the library depending on run-time settings. Enabling this option can degrade performance. Undefined by default.

```
#define QL_NEGATIVE_RATES
```

If defined, negative yield rates are allowed in a few places where they are currently forbidden. It is still not clear whether this is safe. Undefined by default.

```
#define QL_EXTRA_SAFETY_CHECKS
```

If defined, extra run-time checks are added to a few functions. This can prevent their inlining and degrade performance. Undefined by default.

```
#define QL_TODAYS_PAYMENTS
```

If undefined (the default,) payments are considered to be settled at the beginning of the day. Therefore, payments occurring at today's date are not included in the NPV of an instrument.

```
#define QL_DISABLE_DEPRECATED
```

If defined, deprecated code will not be included in the library. Undefined by default.

```
#define QL_USE_INDEXED_COUPON
```

If defined, indexed coupons (see the documentation) are used in floating legs. If undefined (the default), par coupons are used.

```
#define QL_ENABLE_SESSIONS
```

If defined, singletons will return different instances for different sessions. You will have to provide and link with the library a `sessionId()` function in namespace `QuantLib`, returning a different session id for each session.

## 1.6 Usage

To use QuantLib classes in your own code just add

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

at the beginning of your source/header files. Depending on the number of your files in your project, this could cause a large increase in compilation time. If this were not acceptable, collective headers are also available for smaller parts of the library; in particular, each subdirectory of the ql directory contains a file `all.hpp` which makes available all classes and function in the respective subdirectory.

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

### 1.6.1 Microsoft Visual C++

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

1. you won't have to explicitly link your application to the QuantLib library. This is done automatically by compiler directives embedded in the sources.
2. Your project must be compiled with the same options that were used in compiling the QuantLib library you're linking with. For VC6 please

- a) select the appropriate run-time library under project settings, "C/C++" tab, "Code Generation",
- b) check the "Use RTTI" option under the "C++ Language" category.

For VC7 (.NET) under

- a) Property Pages -> C/C++ -> Code Generation -> Runtime Library: please select the appropriate run-time library.
- b) Property Pages -> C/C++ -> Code Generation -> Basic Runtime Checks: please select "Both (/RTC1, equiv. to /RTCsu)".

## 1.7 Frequently asked questions

### Generic questions

- Is it OK to email a QuantLib developer?
- How should I report a bug?
- How can I give back to this project?

### Contributing to the project

- I'm interested in getting involved with the project.
- How do I contribute code to the project?

### Building QuantLib

- I'm having trouble building QuantLib with MinGW.
- I get an error when trying to compile QuantLib in the Dev-C++ IDE.
- I get a compile error about a missing boost header.
- I encounter a linking error about a boost library.
- I encounter a number of compile errors when building the test-suite.
- I'm having trouble building QuantLib with the Sun Studio 11 compiler.

### Testing QuantLib

- The QuantLib test-suite fails under Mac OS X.

### Using QuantLib

- I encounter a linking error under Visual C++.
- QuantLib fails to run correctly under Mac OS X.

### QuantLib features

- Why is feature X missing from QuantLib?

### QuantLib features

- I'm having trouble building/using QuantLibAddin.

### QuantLib mailing lists

- How do I start a new topic?
- How do I prevent my auto-responder from posting to the list?

### QuantLib cross-platform support/extensions

- Does QuantLib support .NET?
- Does QuantLib support FpML? Serialization?

### 1.7.1 Generic questions

#### Is it OK to email a QuantLib developer to ask questions, or seek help, or report a bug?

Yes, it is. However, we urge you to consider posting to the QuantLib mailing list instead. This is for two reasons. The first is that messages on the list are stored: the next one with your problem will be able to find the answer by searching the archives. The second is that you might get your answer sooner. For instance, it just so happens that I am writing this entry in the middle of a two-weeks period without an Internet connection. If anybody wrote me last Monday, the poor soul will wait two weeks for an answer which could have been given already by someone else on the list.

However, if your intent in writing was to call the developer names, disregard the above. By all means write personally to the developer. And possibly, add the line:

```
X-Bogosity: Yes
```

to the mail headers, so that our filters—I mean, WE can take immediate care of it.

#### How should I report a bug?

You can file a bug report using the SourceForge interface at [http://sourceforge.net/tracker/?group\\_id=12740&atid=112740](http://sourceforge.net/tracker/?group_id=12740&atid=112740), or you could write to a QuantLib mailing list.

In any case please report as much details as possible.

If it is a compilation problem please state at least:

- OS system
- compiler (version number, patch level, etc.)
- Boost version
- the compilation error and the file affected

If the test suite fails please report the output obtained by executing the test suite with the following command line options:

```
--log_level=messages --build_info=yes --result_code=no --report_level=short
```

#### Thanks for this project. How can I give back to it?

In true open-source fashion, you can contribute code to the project; see the section '[Contributing to the project](#)' below. This is by far the preferred contribution, closely followed by using the library intensively and reporting any bugs you might find—and possibly patches for fixing them.

However, if you made money by using QuantLib and feel that, as Christmas is getting near, you want to give us a token of your gratitude—well, who am I to discourage you? (for instance, < grin > Luigi's wish list on Amazon UK is at [http://www.amazon.co.uk/exec/obidos/registry/2PC411P4U28CG/ref=w1\\_em\\_to](http://www.amazon.co.uk/exec/obidos/registry/2PC411P4U28CG/ref=w1_em_to), and Nando's is at [http://www.amazon.co.uk/exec/obidos/registry/Q94W7HUR49Z5/ref=w1\\_em\\_to](http://www.amazon.co.uk/exec/obidos/registry/Q94W7HUR49Z5/ref=w1_em_to).)

#### Amazon Wish List? Aren't you ashamed of yourselves?

< broad grin > No, we aren't.

## 1.7.2 Contributing to the project

**I'm interested in getting involved with the project. What should I do?**

Contact the QuantLib group by posting to the developers' mailing list (<[quantlib-dev@lists.sourceforge.net](mailto:quantlib-dev@lists.sourceforge.net)>) and describe your experience and interests. Before doing this, please read:

- the generic introduction for new developers <<http://quantlib.org/newdeveloper.shtml>>
- the project overview <<http://quantlib.org/reference/overview.html>>, with its to-do suggestions
- the Developer FAQ <<http://quantlib.org/developerFAQ.shtml>>
- The Programming Style Guidelines <<http://quantlib.org/style.shtml>>
- the detailed low-level to-do list <<http://quantlib.org/reference/todo.html>>

Also, you might want to specify an area of the library you are particularly interested to, or which would be most useful to you. Asking the administrators to choose a task for you is ok, but it might take time to get an answer and it increases the odds that the chosen task will bore you or otherwise discourage you from completing it.

**How do I contribute code to the project?**

First of all, make sure that contributing code on your part cannot result in litigation about intellectual property. If you work at some financial institution, ask for permission before contributing any relevant portion of code—and get a statement in print.

As for the mechanics of contribution, the preferred way is to submit a patch to the SourceForge patch tracker at <[http://sourceforge.net/tracker/?group\\_id=12740&atid=312740](http://sourceforge.net/tracker/?group_id=12740&atid=312740)>. This will make it less likely that your files are forgotten in the depths of a developer's mailbox.

The preferred format is a diff file as created by the 'patch' utility. If possible, send differences against the CVS repository; diff files based on the latest release might not apply to the latest sources.

If 'patch' is not available on your system or you are not familiar with it, submit the modified files. However, keep in mind that integrating such a contribution will require more work and therefore will take longer.

Finally, contributions should be accompanied by one or more test cases checking the functionality of the new code. While this is not a strict requirement, complying with it will buy from the developers a lot more sympathy towards your contribution.

## 1.7.3 Building QuantLib

**I'm having trouble building QuantLib with MinGW.**

Terry August was kind enough to put together detailed instructions for MinGW users. They can be found at <<http://www.stanford.edu/~taugust/quantlib.html>>.

**When trying to compile QuantLib in the Dev-C++ IDE, I get an error saying that "Could not create makefile: C:\...\Makefile.win"**

The message is misleading. Close the IDE, create an empty sub-directory named "build" in the same directory as the Dev-C++ project, and retry.

Also, a user reported that this was necessary but not sufficient. His workaround was to change the relative paths in Project Options / Build Options to absolute paths (e.g., ".\lib" to "C:\QuantLib-0.3.11\lib").

**When building QuantLib, I get a compile error about a missing boost/something header.**

As mentioned in the readme, QuantLib depends on the Boost library (<http://www.boost.org>). You must download and install it before building QuantLib. After installation, you might have to setup your IDE so that the Boost headers are in its include path.

**When building the test-suite, I encounter a linking error about libboost\_unit\_test\_framework-xxx.**

The folder including the Boost libraries is not in your link path. See the documentation of your compiler for instructions on how to add it.

**But I have no such library on my machine!**

Most likely, you downloaded the Boost distribution and just copied its header files somewhere in your include path. The Boost libraries must be built as well; see [http://www.boost.org/more/getting\\_started.html](http://www.boost.org/more/getting_started.html) for instructions.

**Ok, now I have the library; and the library path is set correctly. But I still cannot link!**

You're using Dev-C++ or MinGW, aren't you? gcc is looking for a library called libboost\_unit\_test\_framework-xxx.a, but the Boost installation process created a libboost\_unit\_test\_framework-xxx.lib instead. Make a copy of the latter in the same location and rename the copy so that it has the correct extension.

**When building the test-suite, I encounter a number of compile errors. I'm using gcc and Boost 1.32.**

This is a Boost problem; you have to apply a one-line patch to your Boost installation. Open boost/test/detail/wrap\_stringstream.hpp and edit line 120,

```
#if !defined(BOOST_NO_STD_LOCALE) && BOOST_WORKAROUND(BOOST_MSVC, >= 1310)
```

so that it reads like:

```
#if !defined(BOOST_NO_STD_LOCALE) && ( !defined(BOOST_MSVC) || BOOST_WORKAROUND(BOOST_MSVC, >= 1310))
```

Also, this problem has been worked around in QuantLib itself since version 0.3.10. You might want to upgrade your QuantLib installation.

**I'm having trouble building QuantLib with the Sun Studio 11 compiler.**

If the error you're getting resembles to

```
>> Assertion:  (../lnk/init.cc, line 1032)
      while processing ../../ql/history.hpp at line 135.
Error code 1
```

you need to patch your compiler. Sun makes the needed patches available at [http://developers.sun.com/prodtech/cc/downloads/patches/ss11\\_patches.html](http://developers.sun.com/prodtech/cc/downloads/patches/ss11_patches.html); you need the ones labeled as "Compilers back-end" and "C++".

## 1.7.4 Testing QuantLib

The QuantLib test-suite fails when compiling under Mac OS X.



We are aware of the problem; apparently, there are issues with global and/or static variables when using shared libraries. As a workaround, compile QuantLib as a static library. This can be accomplished by running configure as:

```
configure --disable-shared
```

### 1.7.5 Using QuantLib

**When linking QuantLib to my project under Visual C++, I encounter the following linking error:**

```
LINK : fatal error LNK1104: cannot open file "QuantLib-vcX-xx-xxx-a_b_c.lib"
```

The folder including QuantLib-vcX-xx-xxx-a\_b\_c.lib is not in your link path (see Project Settings | Link | Input in VC6 or Property Pages | Linker | Input in VC7) or you haven't really built QuantLib-vcX-xx-xxx-a\_b\_c.lib yet. Note that each build configuration produces a different library.

**Programs linking QuantLib fail to run correctly under Mac OS X.**

This is the same problem reported in the '[Testing QuantLib](#)' section; the same workaround applies.

### 1.7.6 QuantLib features

**Why is feature X missing from QuantLib? It would be a very useful one.**

See the section '[Contributing to the project](#)' above.

### 1.7.7 QuantLib features

**I'm having trouble building/using QuantLibAddin.**

The QuantLibAddin project has its own FAQ; see <<http://quantlib.org/quantlibaddin/faq.html>>.

### 1.7.8 QuantLib mailing lists

**How do I start a new topic?**

Use the "New message" command on your mail client; do not reply to another post and change the subject. Although seemingly more convenient for you, replying will cause your message to appear in the wrong thread in the mailing list archive, in the newsgroup interface at Gmane, and in mail clients that support threads. This brings a whole lot of inconvenience on a number of people—possibly including you, as your message might be overlooked by people not interested in the original topic.

**How do I prevent my auto-responder from posting to the list while I'm away?**

It might be possible to configure the auto-responder so that it ignores posts to the mailing list. However, this depends on the particular software you are using.

If that is not possible, you can temporarily prevent the list from posting to your account. It is a bit more of a hassle, but the other list members will appreciate.

Go to your mailing-list configuration page (its direct address is sent to you monthly by SourceForge; alternatively, go to

<<http://lists.sourceforge.net/lists/listinfo/quantlib-users>> and insert your mail address in the last form in the page.)

From there you can enable the "Disable mail delivery" option, causing the posts to the list not to be forwarded to your account.

The mail delivery can be re-enabled later in the same way; you'll have to check the list archives at <<http://sourceforge.net/mailarchive/forum.php?forum=quantlib-users>> to catch up on the posts you missed.

### 1.7.9 QuantLib cross-platform support/extensions

#### Does QuantLib support .NET?

C# has never been officially supported by the QuantLib team. Both <<http://www.quantlib.net>> and <<http://www.capetools.net>> have been "external" attempts which now look like abandoned projects. As such nobody but their authors could help you.

#### Does QuantLib support FpML? Serialization?

Not yet. The subject has been discussed in the mailing lists and a high level design was sketched out:

6-Feb-2005

9-Feb-2005

9-Feb-2005

9-Feb-2005

27-Feb-2005

No further work has been done and help is needed in this area. The next step would be to formalize and document the proposed design, perhaps as a [QuEP](#).

## 1.8 Version history

### Release 0.3.14 - November 2006

#### PORTABILITY

- Version 0.3.14 is the last QuantLib release to support the Borland free compiler 5.5 and Microsoft Visual C++ 6.0. If you use one of these compilers and want support to continue, you can volunteer for maintaining the necessary patches: contact the QuantLib developers for information.

#### LIBOR MARKET MODEL

- This release includes an experimental implementation of a Libor market model developed with Mark Joshi. The interface and its integration with the bulk of the library are still in development.

#### CURRENCIES

- Added Romanian new lev.

#### DATES, CALENDARS, AND DAY COUNTERS

- Added all serial 3M IMM futures (thanks to Toyin Akin.)
- Reworked the Schedule class so that it follows market conventions more closely.
- Added business/252 day-count convention (thanks to Piter Dias.)

#### INTEREST RATES

- Added base swap-rate class and a number of actual swap rates.
- Added constant-maturity swap coupons (including convexity adjustment.)

#### INSTRUMENTS

- Added asset swaps.
- Added face amount to bonds (defaulting to 100.)

#### MATH

- Added hypersphere and lower-diagonal salvaging algorithms (thanks to Yiping Chen.)

#### PRICING ENGINES

- Added Longstaff-Schwartz Monte-Carlo algorithm for American/Bermudan equity options with deterministic interest rates.

#### TERM STRUCTURE

- Added piecewise-spreaded yield curve (thanks to Roland Lichters.)

**Release 0.3.13 - July 31st, 2006**

## CALENDARS

- Added NERC calendar (thanks to Joe Byers.)

## INSTRUMENTS AND PRICING ENGINES

- Added continuous fixed and floating lookback options (thanks to Warren Chou.)
- Added FRA and forward fixed-coupon bonds; examples provided (thanks to Allen Kuo.)
- Added variance swaps (thanks to Warren Chou.)
- Added composite instrument; example provided.
- Added cash-settled swaption pricing in Black swaption engine; test provided.
- Added discrete dividends and soft callability to convertible bonds.

## INTEREST RATES

- Fixed business-day conventions for Euribor and LIBOR indices (following below one month, month-end from one month onwards.)

## MODELS

- Added more complex market parameterizations and performance improvements for Libor market model (thanks to Klaus Spanderen.)

## PROCESSES

- Renamed `BlackScholedProcess` to `GeneralizedBlackScholedProcess`; specialized classes added for Black-Scholes, Merton, Black and Garman-Kohlhagen processes.
- Added Hull-White and G2 processes for Monte Carlo simulation (thanks to Banca Profilo.)

## RANDOM NUMBERS

- Added possibility to skip directly to the n-th item in a Sobol sequence (thanks to Richard Gould.)

## MATH

- Added SABR interpolation for volatilities.
- Added general linear least-squares regression (thanks to Klaus Spanderen.)

**Release 0.3.12 - March 27th, 2006**

## CALENDARS

- Added Brazilian calendar (thanks to Piter Dias.)
- Added Argentinian, Icelandic, Indonesian, Mexican, and Ukrainian calendars.

## INSTRUMENTS AND PRICING ENGINES

- Added convertible bonds (thanks to Theo Boafo.)
- The cash flows returned by the `Bond::cashflows` method now include the redemption.
- `SimpleSwap` can now be set an engine. If none is set, the old cash-flow-based calculation is used.
- Generalized `McVanillaEngine` so that it can manage n-dimensional processes; it now subsumes `McHestonEngine`.
- Added pricing of Bermudan options on binomial trees (thanks to Enrico Michelotti.)
- Separated accrual and payment conventions for bonds.
- Modified basis-point sensitivity calculation so that it returns the cash variation for a basis-point change in rate (it used to return the figure to be multiplied by the variation in order to obtain the same result.)

## MODELS

- Added weights to short-rate model calibration (thanks to Enrico Michelotti.)
- Added Libor market model (thanks to Klaus Spanderen.)

## OPTIMIZATION

- Added Levenberg-Marquardt optimization method (thanks to Klaus Spanderen.)

## EXAMPLES

- Merged American and European option examples; added Bermudan option.
- Added convertible-bond example (thanks to Theo Boafo.)

**Release 0.3.11 - October 20th, 2005**

## GLOBAL FEATURES

- Added configuration option for adding current function information to error messages.
- Added hook for multiple sessions to Singleton.

## CALENDARS

- Added Bombay and Taipei calendars.

## CURRENCIES

- Added new Turkish lira.

## INDEXES

- More accurate LIBOR calendars (thanks to Daniele de Francesco.)

- Added DKKLibor, EURLibor, and NZDLibor indexes.
- Added TRLibor index (thanks to Sercan Atalik.)

#### PRICING ENGINES

- Added Bates stochastic-volatility model; tests provided (thanks to Klaus Spanderen.)
- Added vega to analytic discrete-averaging Asian engine; test provided (thanks to Gary Kennedy.)
- Added stochastic process for caplet Libor market model; tests provided (thanks to Klaus Spanderen.)

#### TERM STRUCTURES

- Added fixed-coupon bond helper for curve bootstrapping (thanks to Toyin Akin.)

#### MATH

- Added tabulated Gauss-Legendre quadratures (thanks to Gary Kennedy.)
- Added more precise implementation of bivariate cumulative normal distribution (thanks to Gary Kennedy.)

#### Release 0.3.10 - July 14th, 2005

#### GLOBAL FEATURES

- The suggested syntax for setting and registering with the global evaluation date is now:

```
Settings::instance().evaluationDate() = date;  
registerWith(Settings::instance().evaluationDate());
```

#### CALENDARS

- Istanbul calendar added (thanks to Serkan Atalik.)

#### LATTICE FRAMEWORK

- Faster implementation of binomial and trinomial trees.

#### MONTE CARLO FRAMEWORK

- Added generic multi-dimensional stochastic process.
- Added stochastic process array (thanks to Klaus Spanderen.)
- Multi-path generator now takes a generic stochastic process; tests provided.
- New Path class implemented which stores asset values rather than variations; this makes pricers independent on whether or not log-variations were calculated. The new class is enabled when `QL_DISABLE_DEPRECATED` is defined; the old class is used otherwise.

#### INSTRUMENTS

- Multi-asset option now takes a generic stochastic process.

## MODELS

- Added Heston stochastic-volatility model; tests provided (thanks to Klaus Spanderen.)  
Provided code include:
  - a corresponding stochastic process;
  - analytic and Monte Carlo option-pricing engines;
  - parameter calibration.

## CASH FLOWS

- Cash-flow analyses such as NPV, IRR, convexity and duration added (thanks to Charles Whitmore.)

## MATH

- Added Gaussian orthogonal polynomials and Gaussian quadratures; tests provided (thanks to Klaus Spanderen.)
- Convergence statistics added; tests provided (thanks to Gary Kennedy.)

## Release 0.3.9 - May 2nd, 2005

### GLOBAL FEATURES

- QL\_SQRT, QL\_MIN etc. deprecated in favor of `std::sqrt`, `std::min`...
- Added a tentative tracing facility to ease debugging.
- Formatters deprecated in favor of output manipulators. A number of data types can now be sent directly to output streams.
- Stream-based implementation of QL\_REQUIRE, QL\_TRACE and similar macros. Together with manipulators, this allows one to write simpler error messages, as in:

```
QL_FAIL("forward at date " << d << " is " << io::rate(f));
```

### INSTRUMENTS

- Improved Bond class
  - yield-related calculation can be performed with either compounded or continuous compounding;
  - added theoretical price based on discount curve;
  - fixed-rate coupon bonds can define different rates for each coupon;
  - added zero-coupon and floating-rate bonds (thanks to StatPro.)
- Option instruments now take a generic StochasticProcess; however, most pricing engines still require a BlackScholesProcess. They should be checked to see whether the requirement can be relaxed. Following this change, Merton76Process no longer inherits from BlackScholesProcess. This avoids erroneous upcasts.

- Partial fix for Bermudan swaptions with exercise lag (thanks to Luca Berardi for the report and discussion.)
- Fix for analytic cap/floor engine; caplets/floorlets whose fixing is in the past are now calculated correctly (thanks to Aurelien Chanudet.)

## CALENDARS

- Added Bratislava and Prague calendars.

## INDICES

- Fixed calendars for LIBOR fixings (thanks to Daniele De Francesco.)

## FINITE\_DIFFERENCES FRAMEWORK

- Migrated finite-difference pricers to pricing-engine framework (thanks to Joseph Wang.)

## YIELD TERM STRUCTURES

- Added generic piecewise yield term structure. Client code can choose what to interpolate (discounts, zero yields, forwards) and how (linear, log-linear, flat) by instantiating types such as:

```
PiecewiseYieldCurve<Discount,LogLinear>
PiecewiseYieldCurve<ZeroYield,Linear>
PiecewiseYieldCurve<ForwardRate,Linear>
```

- Interpolated discount, zero-yield and forward-rate curves can now be set any interpolation.
- FlatForward can now take rates with compounding other than continuous.
- Fix for extrapolation in zero-spreaded and forward-spreaded yield term structure (thanks to Adjriou Belak for the report.)

## MATH

- Added backward- and forward-flat interpolations.

## Release 0.3.8 - December 22nd, 2004

### REQUIRED PACKAGES

- Boost version 1.31.0 or later is now required.

### DOCUMENTATION

- Documentation now includes a [FAQ](#) page.

### GLOBAL FEATURES

- Global evaluation date added through Settings class. Used for index-fixing and exchange-rate lookup.



- added InterestRate class, which encapsulate the interest rate compounding algebra. It manages day-counting convention, compounding convention, conversion between different conventions, and discount/compounding factor calculations. It also has its own formatter.

## INSTRUMENTS

- Bond and FixedCouponBond classes added (thanks to Jeff Yu) providing price/yield conversions; tests provided.

## DATE, CALENDARS, AND DAY COUNT CONVENTIONS

- Reworked Date interface. Added nextWeekday() and nthWeekday() static methods to the class Date. Added nextIMM() for the calculation of the next IMM date.
- Added WeekdayFormatter and FrequencyFormatter
- Added "1/1" day counter. The Actual365 is deprecated: as per ISDA documentation "Actual/365" is the same as "Actual/Actual". Use the ActualActual class instead, or the Actual365Fixed class.
- Added dayCounterFromString(std::string) to QuantLibFunctions.
- Improved Beijing calendar (thanks to Zhou Wu.)

## CURRENCIES AND FX RATES

- Added currency classes; CurrencyTag replaced in library code.
- Added money class providing arithmetic with or without conversions; tests provided.
- Added exchange-rate class; tests provided.
- Added exchange-rate manager with smart rate lookup, i.e., able to derive a missing exchange rate as a chain of provided rates; tests provided.

## MONTE CARLO FRAMEWORK

- Added Faure low-discrepancy sequence (thanks to Gianni Piolanti;) tests provided.
- Added randomized (shifted) low discrepancy sequences that will be used for randomized quasi Monte Carlo.
- Added SeedGenerator class, for random generation of seeds when they are not given by the user.
- Added the implementation of Sobol sequences using the coefficients of the free direction integers as provided by Bratley and Fox, who credited unpublished work of Sobol's and Levitan's.
- Added an implementation of Sobol sequences using the coefficients of the free direction integers of Lemieux, Cieslak, and Luttmer. Coefficients for  $d \leq 40$  are the same as in Bradley-Fox. For dimension  $40 < d \leq 360$  the coefficients have been calculated as optimal values based on the "resolution" criterion. The values has been provided by Christiane Lemieux, private communication, September 2004.
- PathGenerator now works correctly with processes describing  $S$  instead of  $\log S$ . Geometric Brownian process added (thanks to Walter Penschke.)

## LATTICE FRAMEWORK

- Reworked the DiscretizedAsset interface.

## PRICING ENGINES FRAMEWORK

- Added pricing engine for American options with Ju quadratic approximation.
- Average-price Asian pricers have been deprecated. New equivalent pricing engines added.

## FIXED INCOME

- Added current coupon to discretized swap and cap/floor.
- Added IndexManager as a singleton (will replace XiborManager—already obsoleted in library code.)
- Added DayCounter parameter to ParCoupon (to be used for accruing spreads and past fixings.) When missing, it defaults to that of the term structure.
- Added compilation flag to select default floating-coupon type.
- IndexedCoupon can now take a generic index rather than a Libor (thanks to Daniele De Francesco.)
- Added hooks for convexity adjustment in floating-rate coupons; implemented adjustment for InArrearIndexedCoupon.

## YIELD TERM STRUCTURE

- TermStructure renamed to YieldTermStructure (the former name was deprecated.)
- New base class BaseTermStructure which can calculate its reference date based on the global evaluation date. YieldTermStructure, BlackVolTermStructure, LocalVolTermStructure, CapFlatVolatilityStructure, CapletForwardVolatilityStructure, and SwaptionVolatilityStructure are now derived from BaseTermStructure so that they inherit its functionality.

## PATTERNS

- Added Singleton pattern.

## MATH

- Added N-dimensional cubic spline (thanks to Roman Gitlin.)
- Added CovarianceDecomposition class (decomposes a covariance matrix into standard deviations and correlations)

## MISCELLANEA

- Renamed RelinkableHandle to Handle.

## PORTABILITY

- Support for Dev-C++ IDE added.

- Fixes for gcc 2.95 added (thanks to Michael Dirkmann.)

### Release 0.3.7 - July 23rd, 2004

#### IMPORTANT

QuantLib now depends on the Boost library ([www.boost.org](http://www.boost.org)).

You will need a working Boost installation in order to compile and use QuantLib. Instructions for installing Boost from sources are available at <[http://www.boost.org/more/getting\\_started.html](http://www.boost.org/more/getting_started.html)>. Pre-packaged binaries might be available from other sources. Google is your friend (or Debian, or Fink...)

#### DATE, CALENDARS, AND DAY COUNT CONVENTIONS

- Working on differentiating calendars depending on country or exchange, instead of city.
- Added Italy (Settlement, Exchange), United Kingdom (Settlement, Exchange, Metals), United States (Settlement, Exchange, GovernmentBond), Xetra.
- Milan, London, and NewYork calendars have been deprecated.
- Added (old-style) calendars: Beijing, Hong Kong, Riyadh, Seoul, Singapore, Taiwan.
- RollingConvention has been renamed BusinessDayConvention, as for ISDA definitions.

#### MATH

- Added rounding algorithms as per OMG enumeration/definition.

#### TEST SUITE

- Moved to Boost unit test framework. CppUnit is no longer needed.
- Added test for quanto and forward compound engines.
- Added test for roundings.
- Added test for discrete dividend European options.
- Added test for cliquet options.

#### MISCELLANEA

- enable/disableExtrapolation() methods were added to a few classes such as TermStructure. They make it possible to persistently allow extrapolation without the need of specifying it at every method call.
- Added user-configurable flag to disable usage of deprecated classes.

#### PORTABILITY

- Fink package available
- Visual C++ 7.x project files added

**Release 0.3.6 - April 15th, 2004**

Bug-fix release for QuantLib 0.3.5. A bug was removed where calls to `impliedVolatility()` would break the state of the option and of all options sharing the same stochastic process.

**Release 0.3.5 - March 31th, 2004****BOOST SUPPORT**

- When available, QuantLib 0.3.5 now uses parts of the Boost library. The presence of Boost is detected automatically under Unix/Linux systems; on Windows systems, it must be enabled by uncommenting the relevant line in `ql/userconfig.hpp`.
- In the next QuantLib release, the presence of the Boost library will be mandatory.

**MONTE CARLO FRAMEWORK**

- Modified `MultiPath` interface to remove drifts. They are now in the stochastic processes.
- Preliminary implementation of Longstaff-Schwartz least-squares
- Monte Carlo pricer for European basket options
- Brownian-bridge bugs fixed
- `StochasticProcess` base class and derived classes (diffusion, jump-diffusion, etc.) have been created.

**PRICING ENGINES FRAMEWORK**

- Pricing engines now use `Payoff` and `Exercise` classes.
- American basket options.
- Binary barrier option replaced by vanilla option with digital payoff.
- Stulz engine for max and min basket calls and puts on two assets.
- American binary option added (a.k.a. one-touch, american digital, american barrier, etc.) with different payoffs (cash/asset at hit/expiry, etc.)
- Added engine for Merton 1976 jump-diffusion process.
- Added Bjerksund and Stensland approximation for American option (still unstable.)
- Added Barone-Adesi and Whaley approximation for American option.
- Improved Black formula engine with more greeks added.
- Discrete geometric asian option.
- Added Leisen-Reimer binomial tree.

**SHORT RATE MODELS**

- Model renamed to `ShortRateModel`. A typedef is provided for backward compatibility—it will be removed in subsequent releases.

**VOLATILITY FRAMEWORK**

- bug fix for short time ( $0 \leq t \leq T_{\min}$ ) interpolation

#### OPTIMIZATION FRAMEWORK

- Method renamed to `OptimizationMethod`. A typedef is provided for backward compatibility—it will be removed in subsequent releases.

#### PATTERNS

- Composite pattern

#### MATH

- Improved cubic spline interpolation. It now handles end conditions such as first derivative value, second derivative value, not-a-knot. Hyman filter for monotonically constrained interpolation has been implemented. Primitive calculation has been enabled in addition to derivative and second derivative.
- Primitive, first derivative, and second derivative functions are available for linear interpolator.
- Singular value decomposition improved.
- Added bivariate cumulative normal distribution.
- Added binomial coefficient calculation, binomial distribution, cumulative binomial distribution, and Peizer-Pratt inversion (method 2.)
- Added beta functions.
- Added Poisson distribution and cumulative distribution.
- Added incomplete gamma functions.
- Added factorial calculation.
- Added rank-reduced square root and improved pseudo-square root of square symmetric matrices.
- Added Cholesky decomposition.

#### TEST SUITE

- Added test for cubic spline interpolation.
- Added test for singular value decomposition.
- Added test for two-asset baskets using the Stulz pricing engine.
- Added test for Monte Carlo American cash-at-hit options.
- Added test for jump-diffusion engine.
- Added test for American and European digital options.

#### MISCELLANEA

- Inner namespaces have been deprecated.

- Added frequency enumeration, including 'once'.
- MarketElement renamed to Quote.
- Handling strike=0.0 where possible.
- More Payoff classes have been introduced: gap, asset-or-nothing, cash-or-nothing. Payoff is now extensively used.
- Exercise class is now polymorphic. More derived classes have been introduced, and they are now extensively used.
- Introduced QL\_FAIL macro.
- Added calendar for Copenhagen
- 14 April 2004 (election day) added to Johannesburg calendar as a one-off holiday.
- Documentation generated with Doxygen 1.3.6.
- Win32 installer generated with NSIS 2.0.

#### **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 constraints 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-templated 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 of cubic splines.

#### RISK MEASURES

- Introduced semiVariance and regret.
- Redefinition of average shortfall (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.shtml>)

##### VOLATILITY FRAMEWORK

- added Black and local volatility interface

##### PRICING ENGINES FRAMEWORK

- partially implemented QuEP 5 (<http://quantlib.org/quep.shtml>)

##### YIELD TERM STRUCTURE

- interface revisited
- added discrete time forward methods
- added `DiscountCurve` (loglinear interpolated) and `CompoundForward` term structures
- `ForwardSpreadedTermStructure` moved under `QuantLib::TermStructures` 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 calendar 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
- Handle is now castable to an Handle of a compatible type
- added downsideVariance to the Statistics class
- kurtosis() 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

#### **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.shtml>) 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.



## 1.9 Additional resources

The main QuantLib resource is the QuantLib web site (<http://quantlib.org>).

Additional resources available from the above site include:

- current news ([http://sourceforge.net/news/?group\\_id=12740](http://sourceforge.net/news/?group_id=12740));
- the QuantLib mailing lists and forums (<http://quantlib.org/maillinglists.shtml>);
- the QuantLib programming style guidelines (<http://quantlib.org/style.shtml>);
- a link to the QuantLib project page on SourceForge.net (<http://sourceforge.net/projects/quantlib>);
- links to pages for bug reports ([http://sourceforge.net/tracker/?group\\_id=12740&atid=112740](http://sourceforge.net/tracker/?group_id=12740&atid=112740)), patch submissions ([http://sourceforge.net/tracker/?group\\_id=12740&atid=312740](http://sourceforge.net/tracker/?group_id=12740&atid=312740)), and feature requests ([http://sourceforge.net/tracker/?group\\_id=12740&atid=362740](http://sourceforge.net/tracker/?group_id=12740&atid=362740));
- a page (<http://quantlib.org/extensions.shtml>) about how to use QuantLib in other languages/platforms;
- QuantLib web-site statistics ([http://sourceforge.net/project/stats/?group\\_id=12740](http://sourceforge.net/project/stats/?group_id=12740));
- 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, Banca Caboto SpA, administrator
- Luigi Ballabio, StatPro Italia srl, administrator
- Mario Aleppo, StatPro Italia srl
- Marco Bianchetti, Banca Caboto SpA
- Nicolas Di Césaré
- Cristina Duminuco, Banca Caboto SpA
- Dirk Eddelbuettel
- Giorgio Facchinetti, Banca Caboto SpA
- Neil Firth, Mathematical Institute, University of Oxford
- Chiara Fornarola, Banca Caboto SpA
- Nicola Jean, StatPro Italia srl
- Katiuscia Manzoni, Banca Caboto SpA
- Marco Marchioro, StatPro Italia srl
- Mario Pucci, Banca Caboto SpA
- Sadruddin Rejeb
- Enrico Sirola, StatPro Italia srl
- Klaus Spanderen
- Niels Elken Sønderby
- Joseph Wang

### 1.10.2 Contributors

We gratefully acknowledge contributions from Xavier Abulker, Toyin Akin, Sercan Atalik, James Battle, Christopher Baus, Thomas Becker, Adolfo Benin, Luca Berardi, David Binderman, Theo Boafu, Joe Byers, Antoine Cellier, Aurelien Chanudet, Yiping Chen, Warren Chou, Jon Davidson, Daniele De Francesco, Piter Dias, Silvia Frasson, Matteo Gallivanoni, Roman Gitlin, Richard Gould, Tomoya Kawanishi, Gary Kennedy, Allen Kuo, Roland Lichters, André Louw, Enrico Michelotti, Tiziano Müller, Gilbert Pepper, Walter Penschke, Gianni Piolanti, Fabio Ramponi, Peter Schmitteckert, David Schwartz, Eugene Shevkoplyas, Maxim Sokolov, Marco Tarengi, Charles Whitmore, Bernd Johannes Wuebben, and Jeff Yu.

QuantLib also includes code taken from Peter Jäckel's book "Monte Carlo Methods in Finance".

QuantLib includes software developed by the University of Chicago, as Operator of Argonne National Laboratory.

## 1.11 QuantLib License

QuantLib is

Copyright (C) 2000, 2001, 2002, 2003 RiskMap srl  
Copyright (C) 2003, 2004, 2005, 2006 StatPro Italia srl  
Copyright (C) 2002, 2003, 2004, 2005, 2006 Ferdinando Ametrano

Copyright (C) 2001, 2002, 2003 Nicolas Di Césaré  
Copyright (C) 2001, 2002, 2003 Sadruddin Rejeb

Copyright (C) 2002, 2003, 2004 Decillion Pty(Ltd)

Copyright (C) 2003, 2004 Neil Firth  
Copyright (C) 2003, 2004 Roman Gitlin  
Copyright (C) 2003 Niels Elken Søndersby  
Copyright (C) 2003 Kawanishi Tomoya

Copyright (C) 2004 FIMAT Group  
Copyright (C) 2004 M-Dimension Consulting Inc.  
Copyright (C) 2004 Mike Parker  
Copyright (C) 2004 Walter Penschke  
Copyright (C) 2004 Gianni Piolanti  
Copyright (C) 2004, 2005, 2006 Klaus Spanderen  
Copyright (C) 2004 Jeff Yu

Copyright (C) 2005 Toyin Akin  
Copyright (C) 2005 Sercan Atalik  
Copyright (C) 2005, 2006 Theo Boafu  
Copyright (C) 2005, 2006 Piter Dias  
Copyright (C) 2005 Gary Kennedy  
Copyright (C) 2005, 2006 Joseph Wang  
Copyright (C) 2005 Charles Whitmore

Copyright (C) 2006 Banca Profilo S.p.A.  
Copyright (C) 2006 Marco Bianchetti  
Copyright (C) 2006 Yiping Chen  
Copyright (C) 2006 Warren Chou  
Copyright (C) 2006 Cristina Duminuco  
Copyright (C) 2006 Giorgio Facchinetti  
Copyright (C) 2006 Chiara Fornarola  
Copyright (C) 2006 Silvia Frasson  
Copyright (C) 2006 Richard Gould  
Copyright (C) 2006 Mark Joshi  
Copyright (C) 2006 Allen Kuo  
Copyright (C) 2006 Roland Lichters  
Copyright (C) 2006 Katuscia Manzoni  
Copyright (C) 2006 Mario Pucci

QuantLib includes code taken from Peter Jäckel's book "Monte Carlo Methods in Finance".

QuantLib includes software developed by the University of Chicago, as Operator of Argonne National Laboratory.

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 known 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](http://www.opensource.org/docs/certification_mark.html)

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

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

## Chapter 2

# QuantLib Module Index

### 2.1 QuantLib Modules

Here is a list of all modules:

Numeric types . . . . .	101
Currencies and FX rates . . . . .	103
Date and time calculations . . . . .	107
Calendars . . . . .	110
Day counters . . . . .	113
Pricing engines . . . . .	114
Asian option engines . . . . .	115
Barrier option engines . . . . .	116
Basket option engines . . . . .	117
Cap/floor engines . . . . .	118
Cliquet option engines . . . . .	119
Forward option engines . . . . .	120
Quanto option engines . . . . .	121
Swaption engines . . . . .	122
Vanilla option engines . . . . .	123
Finite-differences framework . . . . .	126
Short-rate modelling framework . . . . .	128
Financial instruments . . . . .	131
Lattice methods . . . . .	135
Math tools . . . . .	138
Monte Carlo framework . . . . .	140
Design patterns . . . . .	141
Stochastic processes . . . . .	142
Term structures . . . . .	144
Utilities . . . . .	146
QuantLib macros . . . . .	148
Generic macros . . . . .	149
Numeric limits . . . . .	150
Template capabilities . . . . .	151
Iterator support . . . . .	152
Debugging macros . . . . .	154
Output manipulators . . . . .	153



## Chapter 3

# QuantLib Hierarchical Index

### 3.1 QuantLib Class Hierarchy

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

Abcd	157
AbcdSquared	160
AcyclicVisitor	164
AmericanCondition	167
AmericanPayoffAtExpiry	169
AmericanPayoffAtHit	170
Arguments	185
AssetSwap::arguments	194
CapFloor::arguments	293
Option::arguments	959
VanillaSwap::arguments	1164
VarianceSwap::arguments	1169
Array	187
Average	200
BackwardFlat	201
Barrier	204
BarrierOption::arguments	207
BasketOption::arguments	211
Bicubic	220
Bilinear	222
BinomialDistribution	225
BivariateCumulativeNormalDistributionDr78	229
BivariateCumulativeNormalDistributionWe04DP	230
BlackFormula	235
BlackKarasinski::Dynamics	238
BoundaryCondition	259
DirichletBC	399
NeumannBC	919
BoundaryCondition< QuantLib::TridiagonalOperator >	259
BoxMullerGaussianRng	262
Bridge	266
Calendar	274

Argentina . . . . .	183
Australia . . . . .	199
Brazil . . . . .	263
Canada . . . . .	289
China . . . . .	315
CzechRepublic . . . . .	385
Denmark . . . . .	396
Finland . . . . .	601
Germany . . . . .	698
HongKong . . . . .	711
Hungary . . . . .	722
Iceland . . . . .	723
India . . . . .	741
Indonesia . . . . .	743
Italy . . . . .	776
Japan . . . . .	780
JointCalendar . . . . .	784
Mexico . . . . .	893
NewZealand . . . . .	922
Norway . . . . .	928
NullCalendar . . . . .	931
Poland . . . . .	980
SaudiArabia . . . . .	1018
Singapore . . . . .	1039
Slovakia . . . . .	1048
SouthAfrica . . . . .	1056
SouthKorea . . . . .	1057
Sweden . . . . .	1102
Switzerland . . . . .	1103
Taiwan . . . . .	1106
TARGET . . . . .	1108
Turkey . . . . .	1141
Ukraine . . . . .	1147
UnitedKingdom . . . . .	1149
UnitedStates . . . . .	1151
Constraint . . . . .	338
BoundaryConstraint . . . . .	261
CompositeConstraint . . . . .	329
NoConstraint . . . . .	924
PositiveConstraint . . . . .	981
DayCounter . . . . .	392
Actual360 . . . . .	161
Actual365Fixed . . . . .	162
ActualActual . . . . .	163
Business252 . . . . .	270
OneDayCounter . . . . .	950
SimpleDayCounter . . . . .	1034
Thirty360 . . . . .	1114
Interpolation . . . . .	761
BackwardFlatInterpolation . . . . .	202
CubicSpline . . . . .	373
MonotonicCubicSpline . . . . .	899
NaturalCubicSpline . . . . .	917



NaturalMonotonicCubicSpline . . . . .	918
ForwardFlatInterpolation . . . . .	628
LinearInterpolation . . . . .	817
LogLinearInterpolation . . . . .	840
SABRInterpolation . . . . .	1012
Interpolation2D . . . . .	762
BicubicSpline . . . . .	221
BilinearInterpolation . . . . .	223
Parameter . . . . .	962
ConstantParameter . . . . .	337
NullParameter . . . . .	933
PiecewiseConstantParameter . . . . .	971
TermStructureFittingParameter . . . . .	1112
ExtendedCoxIngersollRoss::FittingParameter . . . . .	586
G2::FittingParameter . . . . .	653
HullWhite::FittingParameter . . . . .	717
Bridge< QuantLib::Calendar, QuantLib::CalendarImpl > . . . . .	266
Bridge< QuantLib::Constraint, QuantLib::ConstraintImpl > . . . . .	266
Bridge< QuantLib::DayCounter, QuantLib::DayCounterImpl > . . . . .	266
Bridge< QuantLib::Interpolation, QuantLib::InterpolationImpl > . . . . .	266
Bridge< QuantLib::Interpolation2D, QuantLib::Interpolation2DImpl > . . . . .	266
Bridge< QuantLib::Parameter, QuantLib::ParameterImpl > . . . . .	266
BrownianBridge . . . . .	268
CalendarImpl . . . . .	282
Calendar::OrthodoxImpl . . . . .	280
Calendar::WesternImpl . . . . .	281
Callability::Price . . . . .	288
Cashflows . . . . .	307
CLGaussianRng . . . . .	316
CliquetOption::arguments . . . . .	319
Clone . . . . .	321
Composite . . . . .	328
ConstantEstimator . . . . .	336
ConstraintImpl . . . . .	339
ContinuousAveragingAsianOption::arguments . . . . .	342
ContinuousFixedLookbackOption::arguments . . . . .	346
ContinuousFloatingLookbackOption::arguments . . . . .	350
ConvergenceStatistics . . . . .	355
ConvertibleBond::option::arguments . . . . .	356
ConvertibleFixedCouponBond . . . . .	358
ConvertibleFloatingRateBond . . . . .	359
ConvertibleZeroCouponBond . . . . .	360
CostFunction . . . . .	362
LeastSquareFunction . . . . .	800
CovarianceDecomposition . . . . .	365
CoxIngersollRoss::Dynamics . . . . .	368
Cubic . . . . .	372
CumulativeBinomialDistribution . . . . .	375
CumulativeNormalDistribution . . . . .	376
CumulativePoissonDistribution . . . . .	377
CuriouslyRecurringTemplate . . . . .	378
Lattice . . . . .	793

Lattice1D . . . . .	795
BlackScholesLattice . . . . .	240
TsiveriotisFernandesLattice . . . . .	1138
OneFactorModel::ShortRateTree . . . . .	954
Lattice2D . . . . .	796
TwoFactorModel::ShortRateTree . . . . .	1145
Solver1D . . . . .	1054
Bisection . . . . .	228
Brent . . . . .	265
FalsePosition . . . . .	591
Newton . . . . .	920
NewtonSafe . . . . .	921
Ridder . . . . .	1007
Secant . . . . .	1021
CuriouslyRecurringTemplate< QuantLib::AdditiveEQPBino	378
Tree< QuantLib::AdditiveEQPBino	1126
BinomialTree< QuantLib::AdditiveEQPBino	226
EqualProbabilitiesBinomialTree< QuantLib::AdditiveEQPBino	437
CuriouslyRecurringTemplate< QuantLib::Bisection > . . . . .	378
Solver1D< QuantLib::Bisection > . . . . .	1054
CuriouslyRecurringTemplate< QuantLib::BlackScholesLattice< T > > . . . . .	378
Lattice< QuantLib::BlackScholesLattice< T > > . . . . .	793
Lattice1D< QuantLib::BlackScholesLattice< T > > . . . . .	795
CuriouslyRecurringTemplate< QuantLib::Brent > . . . . .	378
Solver1D< QuantLib::Brent > . . . . .	1054
CuriouslyRecurringTemplate< QuantLib::CoxRossRubinstein > . . . . .	378
Tree< QuantLib::CoxRossRubinstein > . . . . .	1126
BinomialTree< QuantLib::CoxRossRubinstein > . . . . .	226
EqualJumpsBinomialTree< QuantLib::CoxRossRubinstein > . . . . .	436
CuriouslyRecurringTemplate< QuantLib::FalsePosition > . . . . .	378
Solver1D< QuantLib::FalsePosition > . . . . .	1054
CuriouslyRecurringTemplate< QuantLib::JarrowRudd > . . . . .	378
Tree< QuantLib::JarrowRudd > . . . . .	1126
BinomialTree< QuantLib::JarrowRudd > . . . . .	226
EqualProbabilitiesBinomialTree< QuantLib::JarrowRudd > . . . . .	437
CuriouslyRecurringTemplate< QuantLib::LeisenReimer > . . . . .	378
Tree< QuantLib::LeisenReimer > . . . . .	1126
BinomialTree< QuantLib::LeisenReimer > . . . . .	226
CuriouslyRecurringTemplate< QuantLib::Newton > . . . . .	378
Solver1D< QuantLib::Newton > . . . . .	1054
CuriouslyRecurringTemplate< QuantLib::NewtonSafe > . . . . .	378
Solver1D< QuantLib::NewtonSafe > . . . . .	1054
CuriouslyRecurringTemplate< QuantLib::OneFactorModel::ShortRateTree > . . . . .	378
Lattice< QuantLib::OneFactorModel::ShortRateTree > . . . . .	793
Lattice1D< QuantLib::OneFactorModel::ShortRateTree > . . . . .	795
CuriouslyRecurringTemplate< QuantLib::Ridder > . . . . .	378
Solver1D< QuantLib::Ridder > . . . . .	1054
CuriouslyRecurringTemplate< QuantLib::Secant > . . . . .	378
Solver1D< QuantLib::Secant > . . . . .	1054

CuriouslyRecurringTemplate< QuantLib::Tian > . . . . .	378
Tree< QuantLib::Tian > . . . . .	1126
BinomialTree< QuantLib::Tian > . . . . .	226
CuriouslyRecurringTemplate< QuantLib::Trigeorgis > . . . . .	378
Tree< QuantLib::Trigeorgis > . . . . .	1126
BinomialTree< QuantLib::Trigeorgis > . . . . .	226
EqualJumpsBinomialTree< QuantLib::Trigeorgis > . . . . .	436
CuriouslyRecurringTemplate< QuantLib::TrinomialTree > . . . . .	378
Tree< QuantLib::TrinomialTree > . . . . .	1126
CuriouslyRecurringTemplate< QuantLib::TwoFactorModel::ShortRateTree > . . . . .	378
Lattice< QuantLib::TwoFactorModel::ShortRateTree > . . . . .	793
Lattice2D< QuantLib::TwoFactorModel::ShortRateTree, QuantLib::TrinomialTree > . . . . .	796
CuriouslyRecurringTemplate< T > . . . . .	378
Tree . . . . .	1126
BinomialTree . . . . .	226
EqualJumpsBinomialTree . . . . .	436
CoxRossRubinstein . . . . .	369
Trigeorgis . . . . .	1133
EqualProbabilitiesBinomialTree . . . . .	437
AdditiveEQPBinomialTree . . . . .	165
JarrowRudd . . . . .	782
LeisenReimer . . . . .	803
Tian . . . . .	1115
TrinomialTree . . . . .	1134
Currency . . . . .	379
ARSCurrency . . . . .	190
ATSCurrency . . . . .	196
AUDCurrency . . . . .	197
BDTCurrency . . . . .	216
BEFCurrency . . . . .	217
BGLCurrency . . . . .	219
BRLCurrency . . . . .	267
BYRCurrency . . . . .	271
CADCurrency . . . . .	272
CHFCurrency . . . . .	313
CLPCurrency . . . . .	323
CNYCurrency . . . . .	326
COPCurrency . . . . .	361
CYPCurrency . . . . .	384
CZKCurrency . . . . .	387
DEMCurrency . . . . .	395
DKKCurrency . . . . .	420
EEKCurrency . . . . .	433
ESPCurrency . . . . .	440
EURCurrency . . . . .	443
FIMCurrency . . . . .	599
FRFCurrency . . . . .	649
GBPCurrency . . . . .	679
GRDCurrency . . . . .	701
HKDCurrency . . . . .	710
HUFCurrency . . . . .	713

IEPCurrency . . . . .	725
ILSCurrency . . . . .	726
INRCurrency . . . . .	745
IQDCurrency . . . . .	773
IRRCurrency . . . . .	774
ISKCurrency . . . . .	775
ITLCurrency . . . . .	778
JPYCurrency . . . . .	785
KRWCurrency . . . . .	791
KWDCurrency . . . . .	792
LTLCurrency . . . . .	842
LUFCurrency . . . . .	843
LVLCurrency . . . . .	844
MTLCurrency . . . . .	904
MXNCurrency . . . . .	916
NLGCurrency . . . . .	923
NOKCurrency . . . . .	925
NPRCurrency . . . . .	929
NZDCurrency . . . . .	936
PKRCurrency . . . . .	976
PLNCurrency . . . . .	978
PTECurrency . . . . .	987
ROLCurrency . . . . .	1008
RONCurrency . . . . .	1009
SARCurrency . . . . .	1017
SEKCurrency . . . . .	1024
SGDCurrency . . . . .	1028
SITCurrency . . . . .	1046
SKKCurrency . . . . .	1047
THBCurrency . . . . .	1113
TRLCurrency . . . . .	1135
TRYCurrency . . . . .	1137
TTDCurrency . . . . .	1140
TWDCurrency . . . . .	1142
USDCurrency . . . . .	1156
VEBCurrency . . . . .	1175
ZARCurrency . . . . .	1183
CurveState . . . . .	383
Date . . . . .	388
DayCounterImpl . . . . .	394
Discount . . . . .	400
DiscreteAveragingAsianOption::arguments . . . . .	404
DiscretizedAsset . . . . .	407
DiscretizedDiscountBond . . . . .	410
DiscretizedOption . . . . .	411
Disposable . . . . .	413
DividendVanillaOption::arguments . . . . .	418
DriftCalculator . . . . .	426
Duration . . . . .	429
EarlyExercisePathPricer . . . . .	432
EarlyExercisePathPricer< QuantLib::MultiPath > . . . . .	432
EarlyExercisePathPricer< QuantLib::Path > . . . . .	432
EndCriteria . . . . .	434

ErrorFunction . . . . .	439
Euribor10M . . . . .	445
Euribor11M . . . . .	446
Euribor1M . . . . .	447
Euribor1Y . . . . .	448
Euribor2M . . . . .	449
Euribor2W . . . . .	450
Euribor365_10M . . . . .	452
Euribor365_11M . . . . .	453
Euribor365_1M . . . . .	454
Euribor365_1Y . . . . .	455
Euribor365_2M . . . . .	456
Euribor365_2W . . . . .	457
Euribor365_3M . . . . .	458
Euribor365_3W . . . . .	459
Euribor365_4M . . . . .	460
Euribor365_5M . . . . .	461
Euribor365_6M . . . . .	462
Euribor365_7M . . . . .	463
Euribor365_8M . . . . .	464
Euribor365_9M . . . . .	465
Euribor365_SW . . . . .	466
Euribor3M . . . . .	467
Euribor3W . . . . .	468
Euribor4M . . . . .	469
Euribor5M . . . . .	470
Euribor6M . . . . .	471
Euribor7M . . . . .	472
Euribor8M . . . . .	473
Euribor9M . . . . .	474
EuriborSW . . . . .	475
EURLibor10M . . . . .	509
EURLibor11M . . . . .	510
EURLibor1M . . . . .	511
EURLibor1Y . . . . .	512
EURLibor2M . . . . .	513
EURLibor2W . . . . .	514
EURLibor3M . . . . .	515
EURLibor4M . . . . .	516
EURLibor5M . . . . .	517
EURLibor6M . . . . .	518
EURLibor7M . . . . .	519
EURLibor8M . . . . .	520
EURLibor9M . . . . .	521
EURLiborSW . . . . .	522
EvolutionDescription . . . . .	575
exception	
Error . . . . .	438
ExchangeRate . . . . .	576
Exercise . . . . .	580
EarlyExercise . . . . .	431
AmericanExercise . . . . .	168
BermudanExercise . . . . .	218

EuropeanExercise . . . . .	571
ExtendedCoxIngersollRoss::Dynamics . . . . .	585
Extrapolator . . . . .	589
Interpolation . . . . .	761
Interpolation2D . . . . .	762
TermStructure . . . . .	1109
BlackVolTermStructure . . . . .	252
BlackVarianceTermStructure . . . . .	248
BlackVarianceCurve . . . . .	244
BlackVarianceSurface . . . . .	246
ImpliedVolTermStructure . . . . .	731
BlackVolatilityTermStructure . . . . .	250
BlackConstantVol . . . . .	233
CapletVolatilityStructure . . . . .	299
CapletConstantVolatility . . . . .	297
CapVolatilityStructure . . . . .	301
CapVolatilityVector . . . . .	303
LocalVolTermStructure . . . . .	837
LocalConstantVol . . . . .	830
LocalVolCurve . . . . .	833
LocalVolSurface . . . . .	835
SwaptionVolatilityStructure . . . . .	1099
SwaptionConstantVolatility . . . . .	1090
SwaptionVolatilityCube . . . . .	1093
SwaptionVolatilityCubeBySabr . . . . .	1095
SwaptionVolatilityMatrix . . . . .	1097
YieldTermStructure . . . . .	1179
FlatForward . . . . .	613
ForwardRateStructure . . . . .	639
CompoundForward . . . . .	333
ForwardSpreadedTermStructure . . . . .	641
InterpolatedForwardCurve . . . . .	757
ImpliedTermStructure . . . . .	729
InterpolatedDiscountCurve . . . . .	755
ExtendedDiscountCurve . . . . .	587
InterpolatedDiscountCurve< QuantLib::LogLinear > . . . . .	755
ZeroYieldStructure . . . . .	1189
DriftTermStructure . . . . .	427
InterpolatedZeroCurve . . . . .	759
PiecewiseZeroSpreadedTermStructure . . . . .	974
QuantoTermStructure . . . . .	994
ZeroSpreadedTermStructure . . . . .	1186
Factorial . . . . .	590
FaureRsg . . . . .	592
FDAmericanCondition . . . . .	593
FDDividendEngineMerton73 . . . . .	595
FDDividendEngineShiftScale . . . . .	596
FDEuropeanEngine . . . . .	597
FDStepConditionEngine . . . . .	598
FiniteDifferenceModel . . . . .	600
ForwardFlat . . . . .	627
ForwardOptionArguments . . . . .	631

ForwardRate . . . . .	633
GammaFunction . . . . .	659
Garch11 . . . . .	661
GarmanKlassOpenClose . . . . .	663
GarmanKlassOpenClose< QuantLib::GarmanKlassSigma4 > . . . . .	663
GarmanKlassOpenClose< QuantLib::GarmanKlassSimpleSigma > . . . . .	663
GarmanKlassOpenClose< QuantLib::ParkinsonSigma > . . . . .	663
GaussianOrthogonalPolynomial . . . . .	672
GaussHermitePolynomial . . . . .	669
GaussHyperbolicPolynomial . . . . .	671
GaussJacobiPolynomial . . . . .	675
GaussLaguerrePolynomial . . . . .	677
GaussianQuadrature . . . . .	673
GaussChebyshev2thIntegration . . . . .	665
GaussChebyshevIntegration . . . . .	666
GaussGegenbauerIntegration . . . . .	667
GaussHermiteIntegration . . . . .	668
GaussHyperbolicIntegration . . . . .	670
GaussJacobiIntegration . . . . .	674
GaussLaguerreIntegration . . . . .	676
GaussLegendreIntegration . . . . .	678
GeneralStatistics . . . . .	683
GenericGaussianStatistics . . . . .	688
GenericRiskStatistics . . . . .	692
GenericSequenceStatistics . . . . .	695
DiscrepancyStatistics . . . . .	401
HaltonRsg . . . . .	703
Handle . . . . .	704
HullWhite::Dynamics . . . . .	716
IMM . . . . .	727
IncrementalStatistics . . . . .	735
IntegralEngine . . . . .	749
InterestRate . . . . .	750
Interpolation2DImpl . . . . .	765
Interpolation2D::templateImpl . . . . .	764
InterpolationImpl . . . . .	767
Interpolation::templateImpl . . . . .	766
IntervalPrice . . . . .	768
InverseCumulativeNormal . . . . .	769
InverseCumulativePoisson . . . . .	770
InverseCumulativeRng . . . . .	771
InverseCumulativeRsg . . . . .	772
KnuthUniformRng . . . . .	789
KronrodIntegral . . . . .	790
LeastSquareProblem . . . . .	801
LecuyerUniformRng . . . . .	802
LexicographicalView . . . . .	805
LfmCovarianceParameterization . . . . .	807
LfmCovarianceProxy . . . . .	808
LfmHullWhiteParameterization . . . . .	809
Linear . . . . .	816
LinearLeastSquaresRegression . . . . .	818

LineSearch . . . . .	819
ArmijoLineSearch . . . . .	186
LmCorrelationModel . . . . .	824
LmExponentialCorrelationModel . . . . .	825
LmLinearExponentialCorrelationModel . . . . .	827
LmVolatilityModel . . . . .	829
LmConstWrapperVolatilityModel . . . . .	823
LmLinearExponentialVolatilityModel . . . . .	828
LmExtLinearExponentialVolModel . . . . .	826
LocalVolatilityEstimator . . . . .	832
GarmanKlassAbstract . . . . .	662
SimpleLocalEstimator . . . . .	1035
LocalVolatilityEstimator< QL_REAL > . . . . .	832
LocalVolatilityEstimator< QuantLib::IntervalPrice > . . . . .	832
LogLinear . . . . .	839
MakeMCAmericanEngine . . . . .	845
MakeMCDigitalEngine . . . . .	846
MakeMCEuropeanEngine . . . . .	847
MakeMCEuropeanHestonEngine . . . . .	848
MakeMCHullWhiteCapFloorEngine . . . . .	849
MakeMCVarianceSwapEngine . . . . .	850
MakeSchedule . . . . .	851
MakeVanillaSwap . . . . .	852
map	
TimeBasket . . . . .	1117
MarketModelEvolver . . . . .	855
ForwardRateIpcEvolver . . . . .	637
ForwardRatePcEvolver . . . . .	638
MarketModelMultiProduct . . . . .	856
MarketModelComposite . . . . .	853
MultiProductComposite . . . . .	912
SingleProductComposite . . . . .	1043
MultiProductMultiStep . . . . .	913
MultiProductOneStep . . . . .	914
Matrix . . . . .	857
McPricer . . . . .	884
McCliquetOption . . . . .	867
McDiscreteArithmeticASO . . . . .	870
McEverest . . . . .	876
McHimalaya . . . . .	877
McMaxBasket . . . . .	881
McPagoda . . . . .	882
McPerformanceOption . . . . .	883
McPricer< QuantLib::MultiVariate< QuantLib::GenericPseudoRandomenericPseudo- Random< MersenneTwisterUniformRng, InverseCumulativeNormal > > > .	884
McPricer< QuantLib::SingleVariate< QuantLib::GenericPseudoRandomenericPseudo- Random< MersenneTwisterUniformRng, InverseCumulativeNormal > > > .	884
McSimulation . . . . .	885
MCBarrierEngine . . . . .	863
MCBasketEngine . . . . .	865
MCDiscreteAveragingAsianEngine . . . . .	871



MCDiscreteArithmeticAPEngine . . . . .	869
MCDiscreteGeometricAPEngine . . . . .	873
MCHullWhiteCapFloorEngine . . . . .	878
MCLongstaffSchwartzEngine . . . . .	879
MCAmericanBasketEngine . . . . .	861
MCAmericanEngine . . . . .	862
MCLongstaffSchwartzEngine< QuantLib::BasketOption::engine, QuantLib::Multi-	
Variate< RNG > > . . . . .	879
MCVanillaEngine . . . . .	887
MCDigitalEngine . . . . .	868
MCEuropeanEngine . . . . .	874
MCEuropeanHestonEngine . . . . .	875
MCVarianceSwapEngine . . . . .	888
McSimulation< QuantLib::MultiVariate< RNG >, QuantLib::GenericRiskStatistics > . .	885
McSimulation< QuantLib::MultiVariate< RNG >, S > . . . . .	885
MCVanillaEngine< QuantLib::MultiVariate< RNG >, S > . . . . .	887
McSimulation< QuantLib::SingleVariate< RNG >, S > . . . . .	885
MCLongstaffSchwartzEngine< QuantLib::VanillaOption::engine, QuantLib::Single-	
Variate< RNG >, S > . . . . .	879
MCVanillaEngine< QuantLib::SingleVariate< RNG >, S > . . . . .	887
MersenneTwisterUniformRng . . . . .	890
MixedScheme . . . . .	895
CrankNicolson . . . . .	370
ExplicitEuler . . . . .	581
ImplicitEuler . . . . .	728
Money . . . . .	897
MonteCarloModel . . . . .	900
MoroInverseCumulativeNormal . . . . .	902
MTBrownianGenerator . . . . .	903
MultiAssetOption::arguments . . . . .	907
MultiCubicSpline . . . . .	909
MultiPath . . . . .	910
MultiPathGenerator . . . . .	911
MultiVariate . . . . .	915
NonLinearLeastSquare . . . . .	926
NormalDistribution . . . . .	927
Null . . . . .	930
NumericalMethod . . . . .	934
Lattice . . . . .	793
Lattice< QuantLib::BlackScholesLattice< T > > . . . . .	793
Lattice< QuantLib::OneFactorModel::ShortRateTree > . . . . .	793
Lattice< QuantLib::TwoFactorModel::ShortRateTree > . . . . .	793
Observable . . . . .	938
AffineModel . . . . .	166
G2 . . . . .	651
LiborForwardModel . . . . .	812
OneFactorAffineModel . . . . .	951
CoxIngersollRoss . . . . .	366
ExtendedCoxIngersollRoss . . . . .	583
Vasicek . . . . .	1172
HullWhite . . . . .	714

CalibratedModel . . . . .	283
HestonModel . . . . .	706
BatesModel . . . . .	215
LiborForwardModel . . . . .	812
ShortRateModel . . . . .	1031
OneFactorModel . . . . .	952
BlackKarasinski . . . . .	237
OneFactorAffineModel . . . . .	951
TwoFactorModel . . . . .	1143
G2 . . . . .	651
CalibrationHelper . . . . .	285
CapHelper . . . . .	296
HestonModelHelper . . . . .	707
SwaptionHelper . . . . .	1092
Event . . . . .	573
Callability . . . . .	287
SoftCallability . . . . .	1053
CashFlow . . . . .	305
Coupon . . . . .	363
FixedRateCoupon . . . . .	611
FloatingRateCoupon . . . . .	617
CMSCoupon . . . . .	324
InArrearIndexedCoupon . . . . .	733
ParCoupon . . . . .	964
Short< ParCoupon > . . . . .	1030
UpFrontIndexedCoupon . . . . .	1154
Dividend . . . . .	414
FixedDividend . . . . .	609
FractionalDividend . . . . .	646
SimpleCashFlow . . . . .	1033
Index . . . . .	738
InterestRateIndex . . . . .	753
SwapIndex . . . . .	1081
EuriborSwapFixA . . . . .	476
EuriborSwapFixA10Y . . . . .	477
EuriborSwapFixA12Y . . . . .	478
EuriborSwapFixA15Y . . . . .	479
EuriborSwapFixA1Y . . . . .	480
EuriborSwapFixA20Y . . . . .	481
EuriborSwapFixA25Y . . . . .	482
EuriborSwapFixA2Y . . . . .	483
EuriborSwapFixA30Y . . . . .	484
EuriborSwapFixA3Y . . . . .	485
EuriborSwapFixA4Y . . . . .	486
EuriborSwapFixA5Y . . . . .	487
EuriborSwapFixA6Y . . . . .	488
EuriborSwapFixA7Y . . . . .	489
EuriborSwapFixA8Y . . . . .	490
EuriborSwapFixA9Y . . . . .	491
EuriborSwapFixIFR . . . . .	492
EuriborSwapFixIFR10Y . . . . .	493
EuriborSwapFixIFR12Y . . . . .	494

EuriborSwapFixIFR15Y . . . . .	495
EuriborSwapFixIFR1Y . . . . .	496
EuriborSwapFixIFR20Y . . . . .	497
EuriborSwapFixIFR25Y . . . . .	498
EuriborSwapFixIFR2Y . . . . .	499
EuriborSwapFixIFR30Y . . . . .	500
EuriborSwapFixIFR3Y . . . . .	501
EuriborSwapFixIFR4Y . . . . .	502
EuriborSwapFixIFR5Y . . . . .	503
EuriborSwapFixIFR6Y . . . . .	504
EuriborSwapFixIFR7Y . . . . .	505
EuriborSwapFixIFR8Y . . . . .	506
EuriborSwapFixIFR9Y . . . . .	507
EurliborSwapFixA . . . . .	523
EurliborSwapFixA10Y . . . . .	524
EurliborSwapFixA12Y . . . . .	525
EurliborSwapFixA15Y . . . . .	526
EurliborSwapFixA1Y . . . . .	527
EurliborSwapFixA20Y . . . . .	528
EurliborSwapFixA25Y . . . . .	529
EurliborSwapFixA2Y . . . . .	530
EurliborSwapFixA30Y . . . . .	531
EurliborSwapFixA3Y . . . . .	532
EurliborSwapFixA4Y . . . . .	533
EurliborSwapFixA5Y . . . . .	534
EurliborSwapFixA6Y . . . . .	535
EurliborSwapFixA7Y . . . . .	536
EurliborSwapFixA8Y . . . . .	537
EurliborSwapFixA9Y . . . . .	538
EurliborSwapFixB . . . . .	539
EurliborSwapFixB10Y . . . . .	540
EurliborSwapFixB12Y . . . . .	541
EurliborSwapFixB15Y . . . . .	542
EurliborSwapFixB1Y . . . . .	543
EurliborSwapFixB20Y . . . . .	544
EurliborSwapFixB25Y . . . . .	545
EurliborSwapFixB2Y . . . . .	546
EurliborSwapFixB30Y . . . . .	547
EurliborSwapFixB3Y . . . . .	548
EurliborSwapFixB4Y . . . . .	549
EurliborSwapFixB5Y . . . . .	550
EurliborSwapFixB6Y . . . . .	551
EurliborSwapFixB7Y . . . . .	552
EurliborSwapFixB8Y . . . . .	553
EurliborSwapFixB9Y . . . . .	554
EurliborSwapFixIFR . . . . .	555
EurliborSwapFixIFR10Y . . . . .	556
EurliborSwapFixIFR12Y . . . . .	557
EurliborSwapFixIFR15Y . . . . .	558
EurliborSwapFixIFR1Y . . . . .	559
EurliborSwapFixIFR20Y . . . . .	560
EurliborSwapFixIFR25Y . . . . .	561
EurliborSwapFixIFR2Y . . . . .	562

EurliborSwapFixIFR30Y . . . . .	563
EurliborSwapFixIFR3Y . . . . .	564
EurliborSwapFixIFR4Y . . . . .	565
EurliborSwapFixIFR5Y . . . . .	566
EurliborSwapFixIFR6Y . . . . .	567
EurliborSwapFixIFR7Y . . . . .	568
EurliborSwapFixIFR8Y . . . . .	569
EurliborSwapFixIFR9Y . . . . .	570
Xibor . . . . .	1177
Cdor . . . . .	311
Euribor . . . . .	444
Euribor365 . . . . .	451
Jibar . . . . .	783
Libor . . . . .	811
AUDLibor . . . . .	198
CADLibor . . . . .	273
CHFLibor . . . . .	314
DKKLibor . . . . .	421
EURLibor . . . . .	508
GBPLibor . . . . .	680
JPYLibor . . . . .	786
NZDLibor . . . . .	937
USDLibor . . . . .	1157
Tibor . . . . .	1116
TRLibor . . . . .	1136
Zibor . . . . .	1190
LazyObject . . . . .	798
Instrument . . . . .	746
Bond . . . . .	255
FixedCouponBond . . . . .	602
FloatingRateBond . . . . .	615
ZeroCouponBond . . . . .	1185
CapFloor . . . . .	291
Cap . . . . .	290
Collar . . . . .	327
Floor . . . . .	621
CompositeInstrument . . . . .	330
Forward . . . . .	623
FixedCouponBondForward . . . . .	604
ForwardRateAgreement . . . . .	634
Option . . . . .	958
MultiAssetOption . . . . .	905
BasketOption . . . . .	209
OneAssetOption . . . . .	943
ContinuousFloatingLookbackOption . . . . .	348
OneAssetStrikedOption . . . . .	948
BarrierOption . . . . .	205
CliquetOption . . . . .	317
ContinuousAveragingAsianOption . . . . .	340
ContinuousFixedLookbackOption . . . . .	344
DiscreteAveragingAsianOption . . . . .	402
VanillaOption . . . . .	1160
DividendVanillaOption . . . . .	416

EuropeanOption . . . . .	572
ForwardVanillaOption . . . . .	644
QuantoVanillaOption . . . . .	996
QuantoForwardVanillaOption . . . . .	990
Swaption . . . . .	1085
Stock . . . . .	1074
Swap . . . . .	1079
AssetSwap . . . . .	192
VanillaSwap . . . . .	1162
VarianceSwap . . . . .	1166
PiecewiseYieldCurve . . . . .	972
Link . . . . .	821
PricingEngine . . . . .	983
GenericEngine . . . . .	686
BarrierOption::engine . . . . .	208
AnalyticBarrierEngine . . . . .	171
MCBarrierEngine . . . . .	863
BasketOption::engine . . . . .	212
MCBasketEngine . . . . .	865
MCLongstaffSchwartzEngine< QuantLib::BasketOption::engine, Quant-	
Lib::MultiVariate< RNG > > . . . . .	879
StulzEngine . . . . .	1076
CapFloor::engine . . . . .	294
BlackCapFloorEngine . . . . .	232
MCHullWhiteCapFloorEngine . . . . .	878
CliquetOption::engine . . . . .	320
AnalyticCliquetEngine . . . . .	173
AnalyticPerformanceEngine . . . . .	182
ContinuousAveragingAsianOption::engine . . . . .	343
AnalyticContinuousGeometricAveragePriceAsianEngine . . . . .	176
ContinuousFixedLookbackOption::engine . . . . .	347
AnalyticContinuousFixedLookbackEngine . . . . .	174
ContinuousFloatingLookbackOption::engine . . . . .	351
AnalyticContinuousFloatingLookbackEngine . . . . .	175
ConvertibleBond::option::engine . . . . .	357
BinomialConvertibleEngine . . . . .	224
DiscreteAveragingAsianOption::engine . . . . .	405
AnalyticDiscreteGeometricAveragePriceAsianEngine . . . . .	178
MCDiscreteAveragingAsianEngine . . . . .	871
DividendVanillaOption::engine . . . . .	419
AnalyticDividendEuropeanEngine . . . . .	179
ForwardEngine . . . . .	626
ForwardPerformanceEngine . . . . .	632
GenericModelEngine . . . . .	691
AnalyticCapFloorEngine . . . . .	172
AnalyticHestonEngine . . . . .	181
BatesEngine . . . . .	213
G2SwaptionEngine . . . . .	658
JamshidianSwaptionEngine . . . . .	779
LatticeShortRateModelEngine . . . . .	797
TreeCapFloorEngine . . . . .	1127

TreeSwaptionEngine	1128
TreeVanillaSwapEngine	1129
LatticeShortRateModelEngine< QuantLib::CapFloor::arguments, QuantLib::CapFloor::results >	797
LatticeShortRateModelEngine< QuantLib::Swaption::arguments, QuantLib::Swaption::results >	797
LatticeShortRateModelEngine< QuantLib::VanillaSwap::arguments, QuantLib::VanillaSwap::results >	797
LfmSwaptionEngine	810
MCLongstaffSchwartzEngine	879
QuantoEngine	988
Swaption::engine	1088
BlackSwaptionEngine	243
VarianceSwap::engine	1170
MCVarianceSwapEngine	888
ReplicatingVarianceSwapEngine	1005
GenericEngine< QuantLib::Arguments, QuantLib::Results >	686
GenericModelEngine< QuantLib::ShortRateModel, QuantLib::Arguments, QuantLib::Results >	691
GenericEngine< QuantLib::BarrierOption::arguments, BarrierOption::results >	686
GenericEngine< QuantLib::BasketOption::arguments, BasketOption::results >	686
GenericEngine< QuantLib::CapFloor::arguments, QuantLib::CapFloor::results >	686
GenericModelEngine< QuantLib::AffineModel, QuantLib::CapFloor::arguments, QuantLib::CapFloor::results >	691
GenericModelEngine< QuantLib::ShortRateModel, QuantLib::CapFloor::arguments, QuantLib::CapFloor::results >	691
GenericEngine< QuantLib::CliquetOption::arguments, CliquetOption::results >	686
GenericEngine< QuantLib::ContinuousAveragingAsianOption::arguments, ContinuousAveragingAsianOption::results >	686
GenericEngine< QuantLib::ContinuousFixedLookbackOption::arguments, ContinuousFixedLookbackOption::results >	686
GenericEngine< QuantLib::ContinuousFloatingLookbackOption::arguments, ContinuousFloatingLookbackOption::results >	686
GenericEngine< QuantLib::ConvertibleBond::option::arguments, ConvertibleBond::option::results >	686
GenericEngine< QuantLib::DiscreteAveragingAsianOption::arguments, DiscreteAveragingAsianOption::results >	686
GenericEngine< QuantLib::DividendVanillaOption::arguments, DividendVanillaOption::results >	686
GenericEngine< QuantLib::ForwardOptionArguments< ArgumentsType >, ResultsType >	686
GenericEngine< QuantLib::OneAssetOption::arguments, QuantLib::OneAssetOption::results >	686
GenericEngine< QuantLib::QuantoOptionArguments< ArgumentsType >, QuantLib::QuantoOptionResults< ResultsType > >	686
GenericEngine< QuantLib::Swaption::arguments, QuantLib::Swaption::results >	686
GenericModelEngine< QuantLib::G2, QuantLib::Swaption::arguments, QuantLib::Swaption::results >	691
GenericModelEngine< QuantLib::LiborForwardModel, QuantLib::Swaption::arguments, QuantLib::Swaption::results >	691
GenericModelEngine< QuantLib::OneFactorAffineModel, QuantLib::Swaption::arguments, QuantLib::Swaption::results >	691
GenericModelEngine< QuantLib::ShortRateModel, QuantLib::Swaption::arguments, QuantLib::Swaption::results >	691

GenericEngine< QuantLib::VanillaSwap::arguments, QuantLib::VanillaSwap::results > . . . . .	686
GenericModelEngine< QuantLib::ShortRateModel, QuantLib::VanillaSwap::arguments, QuantLib::VanillaSwap::results > . . . . .	691
GenericEngine< QuantLib::VarianceSwap::arguments, QuantLib::VarianceSwap::results > . . . . .	686
GenericEngine< VanillaOption::arguments, VanillaOption::results > . . . . .	686
GenericModelEngine< QuantLib::HestonModel, VanillaOption::arguments, VanillaOption::results > . . . . .	691
Quote . . . . .	998
CompositeQuote . . . . .	332
DerivedQuote . . . . .	398
SimpleQuote . . . . .	1036
RateHelper . . . . .	1002
FixedCouponBondHelper . . . . .	607
FuturesRateHelper . . . . .	650
RelativeDateRateHelper . . . . .	1004
DepositRateHelper . . . . .	397
FraRateHelper . . . . .	648
SwapRateHelper . . . . .	1083
StochasticProcess . . . . .	1065
ForwardMeasureProcess . . . . .	629
G2ForwardProcess . . . . .	654
G2Process . . . . .	656
HestonProcess . . . . .	708
LiborForwardModelProcess . . . . .	814
StochasticProcess1D . . . . .	1068
ForwardMeasureProcess1D . . . . .	630
HullWhiteForwardProcess . . . . .	718
GeneralizedBlackScholesProcess . . . . .	681
BlackProcess . . . . .	239
BlackScholesMertonProcess . . . . .	241
BlackScholesProcess . . . . .	242
GarmanKohlagenProcess . . . . .	664
GeometricBrownianMotionProcess . . . . .	697
HullWhiteProcess . . . . .	720
Merton76Process . . . . .	891
OrnsteinUhlenbeckProcess . . . . .	960
SquareRootProcess . . . . .	1059
StochasticProcessArray . . . . .	1072
TermStructure . . . . .	1109
TermStructureConsistentModel . . . . .	1111
BlackKarasinski . . . . .	237
ExtendedCoxIngersollRoss . . . . .	583
G2 . . . . .	651
HullWhite . . . . .	714
ObservableValue . . . . .	940
ObservableValue< QuantLib::Date > . . . . .	940
Observer . . . . .	941
BlackCapFloorEngine . . . . .	232
BlackSwaptionEngine . . . . .	243
CalibratedModel . . . . .	283

CalibrationHelper . . . . .	285
CompositeQuote . . . . .	332
DerivedQuote . . . . .	398
FloatingRateCoupon . . . . .	617
GenericModelEngine . . . . .	691
GenericModelEngine< QuantLib::AffineModel, QuantLib::CapFloor::arguments, QuantLib::CapFloor::results > . . . . .	691
GenericModelEngine< QuantLib::G2, QuantLib::Swaption::arguments, Quant- Lib::Swaption::results > . . . . .	691
GenericModelEngine< QuantLib::HestonModel, VanillaOption::arguments, Vanilla- Option::results > . . . . .	691
GenericModelEngine< QuantLib::LiborForwardModel, Quant- Lib::Swaption::arguments, QuantLib::Swaption::results > . . . . .	691
GenericModelEngine< QuantLib::OneFactorAffineModel, Quant- Lib::Swaption::arguments, QuantLib::Swaption::results > . . . . .	691
GenericModelEngine< QuantLib::ShortRateModel, QuantLib::Arguments, Quant- Lib::Results > . . . . .	691
GenericModelEngine< QuantLib::ShortRateModel, QuantLib::CapFloor::arguments, QuantLib::CapFloor::results > . . . . .	691
GenericModelEngine< QuantLib::ShortRateModel, QuantLib::Swaption::arguments, QuantLib::Swaption::results > . . . . .	691
GenericModelEngine< QuantLib::ShortRateModel, QuantLib::Vanilla- Swap::arguments, QuantLib::VanillaSwap::results > . . . . .	691
InterestRateIndex . . . . .	753
LazyObject . . . . .	798
Link . . . . .	821
MCHullWhiteCapFloorEngine . . . . .	878
RateHelper . . . . .	1002
StochasticProcess . . . . .	1065
TermStructure . . . . .	1109
OneAssetOption::arguments . . . . .	946
OneFactorModel::ShortRateDynamics . . . . .	953
OperatorFactory . . . . .	955
OptimizationMethod . . . . .	956
ConjugateGradient . . . . .	335
LevenbergMarquardt . . . . .	804
Simplex . . . . .	1037
SteepestDescent . . . . .	1061
ParameterImpl . . . . .	963
Path . . . . .	965
PathGenerator . . . . .	966
PathPricer . . . . .	967
PathPricer< PathType > . . . . .	967
LongstaffSchwartzPathPricer . . . . .	841
PathPricer< QuantLib::MultiPath > . . . . .	967
PathPricer< QuantLib::Path > . . . . .	967
Payoff . . . . .	968
ForwardTypePayoff . . . . .	643
TypePayoff . . . . .	1146
FloatingTypePayoff . . . . .	620
StrikedTypePayoff . . . . .	1075
AssetOrNothingPayoff . . . . .	191
CashOrNothingPayoff . . . . .	310



GapPayoff . . . . .	660
PercentageStrikePayoff . . . . .	969
PlainVanillaPayoff . . . . .	977
SuperSharePayoff . . . . .	1077
Period . . . . .	970
PoissonDistribution . . . . .	979
PrimeNumbers . . . . .	984
Problem . . . . .	985
QuantoOptionArguments . . . . .	992
QuantoOptionResults . . . . .	993
RandomizedLDS . . . . .	999
RandomSequenceGenerator . . . . .	1001
Results . . . . .	1006
Greeks . . . . .	702
MultiAssetOption::results . . . . .	908
OneAssetOption::results . . . . .	947
MoreGreeks . . . . .	901
OneAssetOption::results . . . . .	947
PriceCurve . . . . .	982
OneAssetOption::results . . . . .	947
Value . . . . .	1158
AssetSwap::results . . . . .	195
CapFloor::results . . . . .	295
MultiAssetOption::results . . . . .	908
OneAssetOption::results . . . . .	947
Swaption::results . . . . .	1089
VanillaSwap::results . . . . .	1165
VarianceSwap::results . . . . .	1171
Rounding . . . . .	1010
CeilingTruncation . . . . .	312
ClosestRounding . . . . .	322
DownRounding . . . . .	423
FloorTruncation . . . . .	622
UpRounding . . . . .	1155
SalvagingAlgorithm . . . . .	1013
Sample . . . . .	1014
SampledCurve . . . . .	1015
Schedule . . . . .	1019
SegmentIntegral . . . . .	1023
Settlement . . . . .	1027
Short . . . . .	1029
ShoutCondition . . . . .	1032
SingleAssetOption . . . . .	1041
DiscreteGeometricASO . . . . .	406
Singleton . . . . .	1044
ExchangeRateManager . . . . .	578
IndexManager . . . . .	740
SeedGenerator . . . . .	1022
Settings . . . . .	1025
Singleton< QuantLib::detail::Tracing > . . . . .	1044
Singleton< QuantLib::ExchangeRateManager > . . . . .	1044
Singleton< QuantLib::IndexManager > . . . . .	1044

Singleton< QuantLib::SeedGenerator > . . . . .	1044
Singleton< QuantLib::Settings > . . . . .	1044
SingleVariate . . . . .	1045
SmileSection . . . . .	1050
SobolRsg . . . . .	1051
StatsHolder . . . . .	1060
step_iterator . . . . .	1062
StepCondition . . . . .	1063
NullCondition . . . . .	932
ZeroCondition . . . . .	1184
StepConditionSet . . . . .	1064
StochasticProcess1D::discretization . . . . .	1070
EulerDiscretization . . . . .	441
StochasticProcess::discretization . . . . .	1071
EulerDiscretization . . . . .	441
SVD . . . . .	1078
Swaption::arguments . . . . .	1087
SymmetricSchurDecomposition . . . . .	1104
TabulatedGaussLegendre . . . . .	1105
TimeGrid . . . . .	1118
TimeSeries . . . . .	1120
TqrEigenDecomposition . . . . .	1122
TransformedGrid . . . . .	1123
TrapezoidIntegral . . . . .	1124
SimpsonIntegral . . . . .	1038
TridiagonalOperator . . . . .	1130
BSMOperator . . . . .	269
DMinus . . . . .	422
DPlus . . . . .	424
DPlusDMinus . . . . .	425
DZero . . . . .	430
TridiagonalOperator::TimeSetter . . . . .	1132
TwoFactorModel::ShortRateDynamics . . . . .	1144
VanillaCMSCouponPricer . . . . .	1159
ConundrumPricer . . . . .	352
ConundrumPricerByNumericalIntegration . . . . .	354
VanillaOption::engine . . . . .	1161
AnalyticDigitalAmericanEngine . . . . .	177
AnalyticEuropeanEngine . . . . .	180
BaroneAdesiWhaleyApproximationEngine . . . . .	203
BinomialVanillaEngine . . . . .	227
Bjerk Sund Stensland Approximation Engine . . . . .	231
FDBermudanEngine . . . . .	594
JumpDiffusionEngine . . . . .	787
JuQuadraticApproximationEngine . . . . .	788
MCLongstaffSchwartzEngine< QuantLib::VanillaOption::engine, QuantLib::Single-	
Variate< RNG >, S > . . . . .	879
MCVanillaEngine . . . . .	887
MCVanillaEngine< QuantLib::MultiVariate< RNG >, S > . . . . .	887
MCVanillaEngine< QuantLib::SingleVariate< RNG >, S > . . . . .	887
Vasicek::Dynamics . . . . .	1174

---

Visitor . . . . .	<a href="#">1176</a>
ZeroYield . . . . .	<a href="#">1188</a>



## Chapter 4

# QuantLib Class Index

### 4.1 QuantLib Class List

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

<a href="#">Abcd</a> ( <a href="#">Abcd</a> functional form for instantaneous volatility ) . . . . .	157
<a href="#">AbcdSquared</a> ( <a href="#">Abcd</a> Squared functional. Helper class ) . . . . .	160
<a href="#">Actual360</a> (Actual/360 day count convention ) . . . . .	161
<a href="#">Actual365Fixed</a> (Actual/365 (Fixed) day count convention ) . . . . .	162
<a href="#">ActualActual</a> (Actual/Actual day count ) . . . . .	163
<a href="#">AcyclicVisitor</a> (Degenerate base class for the Acyclic Visitor pattern ) . . . . .	164
<a href="#">AdditiveEQPBinomialTree</a> (Additive equal probabilities binomial tree ) . . . . .	165
<a href="#">AffineModel</a> (Affine model class ) . . . . .	166
<a href="#">AmericanCondition</a> (American exercise condition ) . . . . .	167
<a href="#">AmericanExercise</a> (American exercise ) . . . . .	168
<a href="#">AmericanPayoffAtExpiry</a> . . . . .	169
<a href="#">AmericanPayoffAtHit</a> . . . . .	170
<a href="#">AnalyticBarrierEngine</a> (Pricing engine for barrier options using analytical formulae ) . .	171
<a href="#">AnalyticCapFloorEngine</a> (Analytic engine for cap/floor ) . . . . .	172
<a href="#">AnalyticCliquetEngine</a> (Pricing engine for Cliquet options using analytical formulae ) .	173
<a href="#">AnalyticContinuousFixedLookbackEngine</a> (Pricing engine for European continuous fixed-strike lookback ) . . . . .	174
<a href="#">AnalyticContinuousFloatingLookbackEngine</a> (Pricing engine for European continuous floating-strike lookback ) . . . . .	175
<a href="#">AnalyticContinuousGeometricAveragePriceAsianEngine</a> (Pricing engine for European continuous geometric average price Asian ) . . . . .	176
<a href="#">AnalyticDigitalAmericanEngine</a> . . . . .	177
<a href="#">AnalyticDiscreteGeometricAveragePriceAsianEngine</a> (Pricing engine for European dis- crete geometric average price Asian ) . . . . .	178
<a href="#">AnalyticDividendEuropeanEngine</a> (Analytic pricing engine for European options with discrete dividends ) . . . . .	179
<a href="#">AnalyticEuropeanEngine</a> (Pricing engine for European vanilla options using analytical formulae ) . . . . .	180
<a href="#">AnalyticHestonEngine</a> (Analytic Heston-model engine based on Fourier transform ) . .	181
<a href="#">AnalyticPerformanceEngine</a> (Pricing engine for performance options using analytical formulae ) . . . . .	182
<a href="#">Argentina</a> (Argentinian calendars ) . . . . .	183
<a href="#">Arguments</a> (Base class for generic argument groups ) . . . . .	185

<a href="#">ArmijoLineSearch</a> (Armijo line search) . . . . .	186
<a href="#">Array</a> (1-D array used in linear algebra) . . . . .	187
<a href="#">ARSCurrency</a> (Argentinian peso) . . . . .	190
<a href="#">AssetOrNothingPayoff</a> (Binary asset-or-nothing payoff) . . . . .	191
<a href="#">AssetSwap</a> (Asset swap) . . . . .	192
<a href="#">AssetSwap::arguments</a> (Arguments for asset swap calculation) . . . . .	194
<a href="#">AssetSwap::results</a> (Results from simple swap calculation) . . . . .	195
<a href="#">ATSCurrency</a> (Austrian shilling) . . . . .	196
<a href="#">AUDCurrency</a> (Australian dollar) . . . . .	197
<a href="#">AUDLibor</a> (AUD LIBOR rate) . . . . .	198
<a href="#">Australia</a> (Australian calendar) . . . . .	199
<a href="#">Average</a> (Placeholder for enumerated averaging types) . . . . .	200
<a href="#">BackwardFlat</a> (Backward-flat interpolation factory and traits) . . . . .	201
<a href="#">BackwardFlatInterpolation</a> (Backward-flat interpolation between discrete points) . . . . .	202
<a href="#">BaroneAdesiWhaleyApproximationEngine</a> . . . . .	203
<a href="#">Barrier</a> (Placeholder for enumerated barrier types) . . . . .	204
<a href="#">BarrierOption</a> (Barrier option on a single asset) . . . . .	205
<a href="#">BarrierOption::arguments</a> (Arguments for barrier option calculation) . . . . .	207
<a href="#">BarrierOption::engine</a> (Barrier engine base class) . . . . .	208
<a href="#">BasketOption</a> (Basket option on a number of assets) . . . . .	209
<a href="#">BasketOption::arguments</a> (Arguments for basket option calculation) . . . . .	211
<a href="#">BasketOption::engine</a> (Basket option engine base class) . . . . .	212
<a href="#">BatesEngine</a> (Bates model engines based on Fourier transform) . . . . .	213
<a href="#">BatesModel</a> . . . . .	215
<a href="#">BDTCurrency</a> (Bangladesh taka) . . . . .	216
<a href="#">BEFCurrency</a> (Belgian franc) . . . . .	217
<a href="#">BermudanExercise</a> (Bermudan exercise) . . . . .	218
<a href="#">BGLCurrency</a> (Bulgarian lev) . . . . .	219
<a href="#">Bicubic</a> (Bicubic-spline interpolation factory) . . . . .	220
<a href="#">BicubicSpline</a> . . . . .	221
<a href="#">Bilinear</a> ( <a href="#">Bilinear</a> interpolation factory) . . . . .	222
<a href="#">BilinearInterpolation</a> ( <a href="#">Bilinear</a> interpolation between discrete points) . . . . .	223
<a href="#">BinomialConvertibleEngine</a> (Binomial Tsiveriotis-Fernandes engine for convertible bonds) . . . . .	224
<a href="#">BinomialDistribution</a> (Binomial probability distribution function) . . . . .	225
<a href="#">BinomialTree</a> (Binomial tree base class) . . . . .	226
<a href="#">BinomialVanillaEngine</a> (Pricing engine for vanilla options using binomial trees) . . . . .	227
<a href="#">Bisection</a> (Bisection 1-D solver) . . . . .	228
<a href="#">BivariateCumulativeNormalDistributionDr78</a> (Cumulative bivariate normal distribution function) . . . . .	229
<a href="#">BivariateCumulativeNormalDistributionWe04DP</a> (Cumulative bivariate normal distribution function (West 2004)) . . . . .	230
<a href="#">BjerkstrandStenslandApproximationEngine</a> . . . . .	231
<a href="#">BlackCapFloorEngine</a> (Black-formula cap/floor engine) . . . . .	232
<a href="#">BlackConstantVol</a> (Constant Black volatility, no time-strike dependence) . . . . .	233
<a href="#">BlackFormula</a> (Black-formula calculator) . . . . .	235
<a href="#">BlackKarasinski</a> (Standard Black-Karasinski model class) . . . . .	237
<a href="#">BlackKarasinski::Dynamics</a> (Short-rate dynamics in the Black-Karasinski model) . . . . .	238
<a href="#">BlackProcess</a> (Black (1976) stochastic process) . . . . .	239
<a href="#">BlackScholesLattice</a> (Simple binomial lattice approximating the Black-Scholes model) . . . . .	240
<a href="#">BlackScholesMertonProcess</a> (Merton (1973) extension to the Black-Scholes stochastic process) . . . . .	241
<a href="#">BlackScholesProcess</a> (Black-Scholes (1973) stochastic process) . . . . .	242
<a href="#">BlackSwaptionEngine</a> (Black-formula swaption engine) . . . . .	243

<a href="#">BlackVarianceCurve</a> (Black volatility curve modelled as variance curve ) . . . . .	244
<a href="#">BlackVarianceSurface</a> (Black volatility surface modelled as variance surface ) . . . . .	246
<a href="#">BlackVarianceTermStructure</a> (Black variance term structure ) . . . . .	248
<a href="#">BlackVolatilityTermStructure</a> (Black-volatility term structure ) . . . . .	250
<a href="#">BlackVolTermStructure</a> (Black-volatility term structure ) . . . . .	252
<a href="#">Bond</a> (Base bond class ) . . . . .	255
<a href="#">BoundaryCondition</a> (Abstract boundary condition class for finite difference problems ) . . . . .	259
<a href="#">BoundaryConstraint</a> (Constraint imposing all arguments to be in [low,high] ) . . . . .	261
<a href="#">BoxMullerGaussianRng</a> (Gaussian random number generator ) . . . . .	262
<a href="#">Brazil</a> (Brazilian calendar ) . . . . .	263
<a href="#">Brent</a> (Brent 1-D solver ) . . . . .	265
<a href="#">Bridge</a> (The Bridge pattern made explicit ) . . . . .	266
<a href="#">BRLCurrency</a> (Brazilian real ) . . . . .	267
<a href="#">BrownianBridge</a> (Builds Wiener process paths using Gaussian variates ) . . . . .	268
<a href="#">BSMOperator</a> (Black-Scholes-Merton differential operator ) . . . . .	269
<a href="#">Business252</a> (Business/252 day count convention ) . . . . .	270
<a href="#">BYRCurrency</a> (Belarussian ruble ) . . . . .	271
<a href="#">CADCurrency</a> (Canadian dollar ) . . . . .	272
<a href="#">CADLibor</a> (CAD LIBOR rate ) . . . . .	273
<a href="#">Calendar</a> (calendar class ) . . . . .	274
<a href="#">Calendar::OrthodoxImpl</a> (Partial calendar implementation ) . . . . .	280
<a href="#">Calendar::WesternImpl</a> (Partial calendar implementation ) . . . . .	281
<a href="#">CalendarImpl</a> (Abstract base class for calendar implementations ) . . . . .	282
<a href="#">CalibratedModel</a> (Calibrated model class ) . . . . .	283
<a href="#">CalibrationHelper</a> (Liquid market instrument used during calibration ) . . . . .	285
<a href="#">Callability</a> ( <a href="#">Instrument</a> callability ) . . . . .	287
<a href="#">Callability::Price</a> (Amount to be paid upon callability ) . . . . .	288
<a href="#">Canada</a> (Canadian calendar ) . . . . .	289
<a href="#">Cap</a> (Concrete cap class ) . . . . .	290
<a href="#">CapFloor</a> (Base class for cap-like instruments ) . . . . .	291
<a href="#">CapFloor::arguments</a> (Arguments for cap/floor calculation ) . . . . .	293
<a href="#">CapFloor::engine</a> (Base class for cap/floor engines ) . . . . .	294
<a href="#">CapFloor::results</a> (Results from cap/floor calculation ) . . . . .	295
<a href="#">CapHelper</a> (Calibration helper for ATM cap ) . . . . .	296
<a href="#">CapletConstantVolatility</a> (Constant caplet volatility, no time-strike dependence ) . . . . .	297
<a href="#">CapletVolatilityStructure</a> (Caplet/floorlet forward-volatility structure ) . . . . .	299
<a href="#">CapVolatilityStructure</a> (Cap/floor term-volatility structure ) . . . . .	301
<a href="#">CapVolatilityVector</a> (Cap/floor at-the-money term-volatility vector ) . . . . .	303
<a href="#">CashFlow</a> (Base class for cash flows ) . . . . .	305
<a href="#">Cashflows</a> ( <a href="#">Cashflows</a> analysis functions ) . . . . .	307
<a href="#">CashOrNothingPayoff</a> (Binary cash-or-nothing payoff ) . . . . .	310
<a href="#">Cdor</a> (CDOR rate ) . . . . .	311
<a href="#">CeilingTruncation</a> (Ceiling truncation ) . . . . .	312
<a href="#">CHFCurrency</a> (Swiss franc ) . . . . .	313
<a href="#">CHFLibor</a> (CHF LIBOR rate ) . . . . .	314
<a href="#">China</a> (Chinese calendar ) . . . . .	315
<a href="#">CLGaussianRng</a> (Gaussian random number generator ) . . . . .	316
<a href="#">CliquetOption</a> (Cliquet (Ratchet) option ) . . . . .	317
<a href="#">CliquetOption::arguments</a> (Arguments for cliquet option calculation ) . . . . .	319
<a href="#">CliquetOption::engine</a> (Cliquet engine base class ) . . . . .	320
<a href="#">Clone</a> (Cloning proxy to an underlying object ) . . . . .	321
<a href="#">ClosestRounding</a> (Closest rounding ) . . . . .	322
<a href="#">CLPCurrency</a> (Chilean peso ) . . . . .	323
<a href="#">CMSCoupon</a> (CMS coupon class ) . . . . .	324

CNYCurrency (Chinese yuan)	326
Collar (Concrete collar class)	327
Composite (Composite pattern)	328
CompositeConstraint (Constraint enforcing both given sub-constraints)	329
CompositeInstrument (Composite instrument)	330
CompositeQuote (Market element whose value depends on two other market element)	332
CompoundForward (Compound-forward structure)	333
ConjugateGradient (Multi-dimensional Conjugate Gradient class)	335
ConstantEstimator	336
ConstantParameter (Standard constant parameter $a(t) = a$ )	337
Constraint (Base constraint class)	338
ConstraintImpl (Base class for constraint implementations)	339
ContinuousAveragingAsianOption (Continuous-averaging Asian option)	340
ContinuousAveragingAsianOption::arguments (Extra arguments for single-asset continuous-average Asian option)	342
ContinuousAveragingAsianOption::engine (Continuous-averaging Asian engine base class)	343
ContinuousFixedLookbackOption (Continuous-fixed lookback option)	344
ContinuousFixedLookbackOption::arguments (Arguments for continuous fixed lookback option calculation)	346
ContinuousFixedLookbackOption::engine (Continuous fixed lookback engine base class)	347
ContinuousFloatingLookbackOption (Continuous-floating lookback option)	348
ContinuousFloatingLookbackOption::arguments (Arguments for continuous floating lookback option calculation)	350
ContinuousFloatingLookbackOption::engine (Continuous floating lookback engine base class)	351
ConundrumPricer (ConundrumPricer)	352
ConundrumPricerByNumericalIntegration (ConundrumPricerByNumericalIntegration)	354
ConvergenceStatistics (Statistics class with convergence table)	355
ConvertibleBond::option::arguments (Arguments for Convertible Bond calculation)	356
ConvertibleBond::option::engine (Convertible bond engine base class)	357
ConvertibleFixedCouponBond (Convertible fixed-coupon bond)	358
ConvertibleFloatingRateBond (Convertible floating-rate bond)	359
ConvertibleZeroCouponBond (Convertible zero-coupon bond)	360
COPCurrency (Colombian peso)	361
CostFunction (Cost function abstract class for optimization problem)	362
Coupon (coupon accruing over a fixed period)	363
CovarianceDecomposition	365
CoxIngersollRoss (Cox-Ingersoll-Ross model class)	366
CoxIngersollRoss::Dynamics (Dynamics of the short-rate under the Cox-Ingersoll-Ross model)	368
CoxRossRubinstein (Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree)	369
CrankNicolson (Crank-Nicolson scheme for finite difference methods)	370
Cubic (cubic-spline interpolation factory and traits)	372
CubicSpline (Cubic spline interpolation between discrete points)	373
CumulativeBinomialDistribution (Cumulative binomial distribution function)	375
CumulativeNormalDistribution (Cumulative normal distribution function)	376
CumulativePoissonDistribution (Cumulative Poisson distribution function)	377
CuriouslyRecurringTemplate (Support for the curiously recurring template pattern)	378
Currency (Currency specification)	379
CurveState	383
CYPCurrency (Cyprus pound)	384
CzechRepublic (Czech calendars)	385
CZKCurrency (Czech koruna)	387



Date (Concrete date class ) . . . . .	388
DayCounter (Day counter class ) . . . . .	392
DayCounterImpl (Abstract base class for day counter implementations ) . . . . .	394
DEMCurrency (Deutsche mark ) . . . . .	395
Denmark (Danish calendar ) . . . . .	396
DepositRateHelper (Rate helper for bootstrapping over deposit rates ) . . . . .	397
DerivedQuote (Market element whose value depends on another market element ) . . . . .	398
DirichletBC (Neumann boundary condition (i.e., constant value) ) . . . . .	399
Discount (Discount-curve traits ) . . . . .	400
DiscrepancyStatistics (Statistic tool for sequences with discrepancy calculation ) . . . . .	401
DiscreteAveragingAsianOption (Discrete-averaging Asian option ) . . . . .	402
DiscreteAveragingAsianOption::arguments (Extra arguments for single-asset discrete-average Asian option ) . . . . .	404
DiscreteAveragingAsianOption::engine (Discrete-averaging Asian engine base class ) . . . . .	405
DiscreteGeometricASO (Discrete geometric average-strike Asian option (European style) ) . . . . .	406
DiscretizedAsset (Discretized asset class used by numerical methods ) . . . . .	407
DiscretizedDiscountBond (Useful discretized discount bond asset ) . . . . .	410
DiscretizedOption (Discretized option on a given asset ) . . . . .	411
Disposable (Generic disposable object with move semantics ) . . . . .	413
Dividend (Predetermined cash flow ) . . . . .	414
DividendVanillaOption (Single-asset vanilla option (no barriers) with discrete dividends ) . . . . .	416
DividendVanillaOption::arguments (Arguments for dividend vanilla option calculation ) . . . . .	418
DividendVanillaOption::engine (Dividend vanilla option engine base class ) . . . . .	419
DKKCurrency (Danish krone ) . . . . .	420
DKKLibor (DKK LIBOR rate ) . . . . .	421
DMinus ( $D_-$ matricial representation ) . . . . .	422
DownRounding (Down-rounding ) . . . . .	423
DPlus ( $D_+$ matricial representation ) . . . . .	424
DPlusDMinus ( $D_+D_-$ matricial representation ) . . . . .	425
DriftCalculator (Drift computation for Market Models ) . . . . .	426
DriftTermStructure (Drift term structure ) . . . . .	427
Duration (Duration type ) . . . . .	429
DZero ( $D_0$ matricial representation ) . . . . .	430
EarlyExercise (Early-exercise base class ) . . . . .	431
EarlyExercisePathPricer (Base class for early exercise path pricers ) . . . . .	432
EEKCurrency (Estonian kroon ) . . . . .	433
EndCriteria (Criteria to end optimization process ) . . . . .	434
EqualJumpsBinomialTree (Base class for equal jumps binomial tree ) . . . . .	436
EqualProbabilitiesBinomialTree (Base class for equal probabilities binomial tree ) . . . . .	437
Error (Base error class ) . . . . .	438
ErrorFunction (Error function ) . . . . .	439
ESPCurrency (Spanish peseta ) . . . . .	440
EulerDiscretization (Euler discretization for stochastic processes ) . . . . .	441
EURCurrency (European Euro ) . . . . .	443
Euribor (Euribor index ) . . . . .	444
Euribor10M (10-months Euribor index ) . . . . .	445
Euribor11M (11-months Euribor index ) . . . . .	446
Euribor1M (1-month Euribor index ) . . . . .	447
Euribor1Y (1-year Euribor index ) . . . . .	448
Euribor2M (2-months Euribor index ) . . . . .	449
Euribor2W (2-weeks Euribor index ) . . . . .	450
Euribor365 (Actual/365 Euribor index ) . . . . .	451
Euribor365_10M (10-months Euribor365 index ) . . . . .	452
Euribor365_11M (11-months Euribor365 index ) . . . . .	453

<a href="#">Euribor365_1M</a> (1-month Euribor365 index ) . . . . .	454
<a href="#">Euribor365_1Y</a> (1-year Euribor365 index ) . . . . .	455
<a href="#">Euribor365_2M</a> (2-months Euribor365 index ) . . . . .	456
<a href="#">Euribor365_2W</a> (2-weeks Euribor365 index ) . . . . .	457
<a href="#">Euribor365_3M</a> (3-months Euribor365 index ) . . . . .	458
<a href="#">Euribor365_3W</a> (3-weeks Euribor365 index ) . . . . .	459
<a href="#">Euribor365_4M</a> (4-months Euribor365 index ) . . . . .	460
<a href="#">Euribor365_5M</a> (5-months Euribor365 index ) . . . . .	461
<a href="#">Euribor365_6M</a> (6-months Euribor365 index ) . . . . .	462
<a href="#">Euribor365_7M</a> (7-months Euribor365 index ) . . . . .	463
<a href="#">Euribor365_8M</a> (8-months Euribor365 index ) . . . . .	464
<a href="#">Euribor365_9M</a> (9-months Euribor365 index ) . . . . .	465
<a href="#">Euribor365_SW</a> (1-week Euribor365 index ) . . . . .	466
<a href="#">Euribor3M</a> (3-months Euribor index ) . . . . .	467
<a href="#">Euribor3W</a> (3-weeks Euribor index ) . . . . .	468
<a href="#">Euribor4M</a> (4-months Euribor index ) . . . . .	469
<a href="#">Euribor5M</a> (5-months Euribor index ) . . . . .	470
<a href="#">Euribor6M</a> (6-months Euribor index ) . . . . .	471
<a href="#">Euribor7M</a> (7-months Euribor index ) . . . . .	472
<a href="#">Euribor8M</a> (8-months Euribor index ) . . . . .	473
<a href="#">Euribor9M</a> (9-months Euribor index ) . . . . .	474
<a href="#">EuriborSW</a> (1-week Euribor index ) . . . . .	475
<a href="#">EuriborSwapFixA</a> (EuriborSwapFixA index ) . . . . .	476
<a href="#">EuriborSwapFixA10Y</a> (10-year EuriborSwapFixA index ) . . . . .	477
<a href="#">EuriborSwapFixA12Y</a> (12-year EuriborSwapFixA index ) . . . . .	478
<a href="#">EuriborSwapFixA15Y</a> (15-year EuriborSwapFixA index ) . . . . .	479
<a href="#">EuriborSwapFixA1Y</a> (1-year EuriborSwapFixA index ) . . . . .	480
<a href="#">EuriborSwapFixA20Y</a> (20-year EuriborSwapFixA index ) . . . . .	481
<a href="#">EuriborSwapFixA25Y</a> (25-year EuriborSwapFixA index ) . . . . .	482
<a href="#">EuriborSwapFixA2Y</a> (2-year EuriborSwapFixA index ) . . . . .	483
<a href="#">EuriborSwapFixA30Y</a> (30-year EuriborSwapFixA index ) . . . . .	484
<a href="#">EuriborSwapFixA3Y</a> (3-year EuriborSwapFixA index ) . . . . .	485
<a href="#">EuriborSwapFixA4Y</a> (4-year EuriborSwapFixA index ) . . . . .	486
<a href="#">EuriborSwapFixA5Y</a> (5-year EuriborSwapFixA index ) . . . . .	487
<a href="#">EuriborSwapFixA6Y</a> (6-year EuriborSwapFixA index ) . . . . .	488
<a href="#">EuriborSwapFixA7Y</a> (7-year EuriborSwapFixA index ) . . . . .	489
<a href="#">EuriborSwapFixA8Y</a> (8-year EuriborSwapFixA index ) . . . . .	490
<a href="#">EuriborSwapFixA9Y</a> (9-year EuriborSwapFixA index ) . . . . .	491
<a href="#">EuriborSwapFixIFR</a> (EuriborSwapFixIFR index ) . . . . .	492
<a href="#">EuriborSwapFixIFR10Y</a> (10-year EuriborSwapFixIFR index ) . . . . .	493
<a href="#">EuriborSwapFixIFR12Y</a> (12-year EuriborSwapFixIFR index ) . . . . .	494
<a href="#">EuriborSwapFixIFR15Y</a> (15-year EuriborSwapFixIFR index ) . . . . .	495
<a href="#">EuriborSwapFixIFR1Y</a> (1-year EuriborSwapFixIFR index ) . . . . .	496
<a href="#">EuriborSwapFixIFR20Y</a> (20-year EuriborSwapFixIFR index ) . . . . .	497
<a href="#">EuriborSwapFixIFR25Y</a> (25-year EuriborSwapFixIFR index ) . . . . .	498
<a href="#">EuriborSwapFixIFR2Y</a> (2-year EuriborSwapFixIFR index ) . . . . .	499
<a href="#">EuriborSwapFixIFR30Y</a> (30-year EuriborSwapFixIFR index ) . . . . .	500
<a href="#">EuriborSwapFixIFR3Y</a> (3-year EuriborSwapFixIFR index ) . . . . .	501
<a href="#">EuriborSwapFixIFR4Y</a> (4-year EuriborSwapFixIFR index ) . . . . .	502
<a href="#">EuriborSwapFixIFR5Y</a> (5-year EuriborSwapFixIFR index ) . . . . .	503
<a href="#">EuriborSwapFixIFR6Y</a> (6-year EuriborSwapFixIFR index ) . . . . .	504
<a href="#">EuriborSwapFixIFR7Y</a> (7-year EuriborSwapFixIFR index ) . . . . .	505
<a href="#">EuriborSwapFixIFR8Y</a> (8-year EuriborSwapFixIFR index ) . . . . .	506
<a href="#">EuriborSwapFixIFR9Y</a> (9-year EuriborSwapFixIFR index ) . . . . .	507

<a href="#">EURLibor</a> (EUR LIBOR rate ) . . . . .	508
<a href="#">EURLibor10M</a> (10-months EURLibor index ) . . . . .	509
<a href="#">EURLibor11M</a> (11-months EURLibor index ) . . . . .	510
<a href="#">EURLibor1M</a> (1-month EURLibor index ) . . . . .	511
<a href="#">EURLibor1Y</a> (1-year EURLibor index ) . . . . .	512
<a href="#">EURLibor2M</a> (2-months EURLibor index ) . . . . .	513
<a href="#">EURLibor2W</a> (2-weeks Euribor index ) . . . . .	514
<a href="#">EURLibor3M</a> (3-months EURLibor index ) . . . . .	515
<a href="#">EURLibor4M</a> (4-months EURLibor index ) . . . . .	516
<a href="#">EURLibor5M</a> (5-months EURLibor index ) . . . . .	517
<a href="#">EURLibor6M</a> (6-months EURLibor index ) . . . . .	518
<a href="#">EURLibor7M</a> (7-months EURLibor index ) . . . . .	519
<a href="#">EURLibor8M</a> (8-months EURLibor index ) . . . . .	520
<a href="#">EURLibor9M</a> (9-months EURLibor index ) . . . . .	521
<a href="#">EURLiborSW</a> (1-week EURLibor index ) . . . . .	522
<a href="#">EurliborSwapFixA</a> (EurliborSwapFixA index ) . . . . .	523
<a href="#">EurliborSwapFixA10Y</a> (10-year EurliborSwapFixA index ) . . . . .	524
<a href="#">EurliborSwapFixA12Y</a> (12-year EurliborSwapFixA index ) . . . . .	525
<a href="#">EurliborSwapFixA15Y</a> (15-year EurliborSwapFixA index ) . . . . .	526
<a href="#">EurliborSwapFixA1Y</a> (1-year EurliborSwapFixA index ) . . . . .	527
<a href="#">EurliborSwapFixA20Y</a> (20-year EurliborSwapFixA index ) . . . . .	528
<a href="#">EurliborSwapFixA25Y</a> (25-year EurliborSwapFixA index ) . . . . .	529
<a href="#">EurliborSwapFixA2Y</a> (2-year EurliborSwapFixA index ) . . . . .	530
<a href="#">EurliborSwapFixA30Y</a> (30-year EurliborSwapFixA index ) . . . . .	531
<a href="#">EurliborSwapFixA3Y</a> (3-year EurliborSwapFixA index ) . . . . .	532
<a href="#">EurliborSwapFixA4Y</a> (4-year EurliborSwapFixA index ) . . . . .	533
<a href="#">EurliborSwapFixA5Y</a> (5-year EurliborSwapFixA index ) . . . . .	534
<a href="#">EurliborSwapFixA6Y</a> (6-year EurliborSwapFixA index ) . . . . .	535
<a href="#">EurliborSwapFixA7Y</a> (7-year EurliborSwapFixA index ) . . . . .	536
<a href="#">EurliborSwapFixA8Y</a> (8-year EurliborSwapFixA index ) . . . . .	537
<a href="#">EurliborSwapFixA9Y</a> (9-year EurliborSwapFixA index ) . . . . .	538
<a href="#">EurliborSwapFixB</a> (EurliborSwapFixB index ) . . . . .	539
<a href="#">EurliborSwapFixB10Y</a> (10-year EurliborSwapFixB index ) . . . . .	540
<a href="#">EurliborSwapFixB12Y</a> (12-year EurliborSwapFixB index ) . . . . .	541
<a href="#">EurliborSwapFixB15Y</a> (15-year EurliborSwapFixB index ) . . . . .	542
<a href="#">EurliborSwapFixB1Y</a> (1-year EurliborSwapFixB index ) . . . . .	543
<a href="#">EurliborSwapFixB20Y</a> (20-year EurliborSwapFixB index ) . . . . .	544
<a href="#">EurliborSwapFixB25Y</a> (25-year EurliborSwapFixB index ) . . . . .	545
<a href="#">EurliborSwapFixB2Y</a> (2-year EurliborSwapFixB index ) . . . . .	546
<a href="#">EurliborSwapFixB30Y</a> (30-year EurliborSwapFixB index ) . . . . .	547
<a href="#">EurliborSwapFixB3Y</a> (3-year EurliborSwapFixB index ) . . . . .	548
<a href="#">EurliborSwapFixB4Y</a> (4-year EurliborSwapFixB index ) . . . . .	549
<a href="#">EurliborSwapFixB5Y</a> (5-year EurliborSwapFixB index ) . . . . .	550
<a href="#">EurliborSwapFixB6Y</a> (6-year EurliborSwapFixB index ) . . . . .	551
<a href="#">EurliborSwapFixB7Y</a> (7-year EurliborSwapFixB index ) . . . . .	552
<a href="#">EurliborSwapFixB8Y</a> (8-year EurliborSwapFixB index ) . . . . .	553
<a href="#">EurliborSwapFixB9Y</a> (9-year EurliborSwapFixB index ) . . . . .	554
<a href="#">EurliborSwapFixIFR</a> (EurliborSwapFixIFR index ) . . . . .	555
<a href="#">EurliborSwapFixIFR10Y</a> (10-year EurliborSwapFixIFR index ) . . . . .	556
<a href="#">EurliborSwapFixIFR12Y</a> (12-year EurliborSwapFixIFR index ) . . . . .	557
<a href="#">EurliborSwapFixIFR15Y</a> (15-year EurliborSwapFixIFR index ) . . . . .	558
<a href="#">EurliborSwapFixIFR1Y</a> (1-year EurliborSwapFixIFR index ) . . . . .	559
<a href="#">EurliborSwapFixIFR20Y</a> (20-year EurliborSwapFixIFR index ) . . . . .	560
<a href="#">EurliborSwapFixIFR25Y</a> (25-year EurliborSwapFixIFR index ) . . . . .	561

<a href="#">EurliborSwapFixIFR2Y</a> (2-year EurliborSwapFixIFR index ) . . . . .	562
<a href="#">EurliborSwapFixIFR30Y</a> (30-year EurliborSwapFixIFR index ) . . . . .	563
<a href="#">EurliborSwapFixIFR3Y</a> (3-year EurliborSwapFixIFR index ) . . . . .	564
<a href="#">EurliborSwapFixIFR4Y</a> (4-year EurliborSwapFixIFR index ) . . . . .	565
<a href="#">EurliborSwapFixIFR5Y</a> (5-year EurliborSwapFixIFR index ) . . . . .	566
<a href="#">EurliborSwapFixIFR6Y</a> (6-year EurliborSwapFixIFR index ) . . . . .	567
<a href="#">EurliborSwapFixIFR7Y</a> (7-year EurliborSwapFixIFR index ) . . . . .	568
<a href="#">EurliborSwapFixIFR8Y</a> (8-year EurliborSwapFixIFR index ) . . . . .	569
<a href="#">EurliborSwapFixIFR9Y</a> (9-year EurliborSwapFixIFR index ) . . . . .	570
<a href="#">EuropeanExercise</a> (European exercise ) . . . . .	571
<a href="#">EuropeanOption</a> (European option on a single asset ) . . . . .	572
<a href="#">Event</a> (Base class for event ) . . . . .	573
<a href="#">EvolutionDescription</a> . . . . .	575
<a href="#">ExchangeRate</a> (Exchange rate between two currencies ) . . . . .	576
<a href="#">ExchangeRateManager</a> (Exchange-rate repository ) . . . . .	578
<a href="#">Exercise</a> (Base exercise class ) . . . . .	580
<a href="#">ExplicitEuler</a> ( <a href="#">Forward</a> Euler scheme for finite difference methods ) . . . . .	581
<a href="#">ExtendedCoxIngersollRoss</a> (Extended Cox-Ingersoll-Ross model class ) . . . . .	583
<a href="#">ExtendedCoxIngersollRoss::Dynamics</a> (Short-rate dynamics in the extended Cox-Ingersoll-Ross model ) . . . . .	585
<a href="#">ExtendedCoxIngersollRoss::FittingParameter</a> (Analytical term-structure fitting parameter $\varphi(t)$ ) . . . . .	586
<a href="#">ExtendedDiscountCurve</a> (Term structure based on loglinear interpolation of discount factors ) . . . . .	587
<a href="#">Extrapolator</a> (Base class for classes possibly allowing extrapolation ) . . . . .	589
<a href="#">Factorial</a> (Factorial numbers calculator ) . . . . .	590
<a href="#">FalsePosition</a> (False position 1-D solver ) . . . . .	591
<a href="#">FaureRsg</a> (Faure low-discrepancy sequence generator ) . . . . .	592
<a href="#">FDAmericanCondition</a> . . . . .	593
<a href="#">FDBermudanEngine</a> (Finite-differences Bermudan engine ) . . . . .	594
<a href="#">FDDividendEngineMerton73</a> (Finite-differences pricing engine for dividend options using ) . . . . .	595
<a href="#">FDDividendEngineShiftScale</a> (Finite-differences pricing engine for dividend options using ) . . . . .	596
<a href="#">FDEuropeanEngine</a> (Pricing engine for European options using finite-differences ) . . . . .	597
<a href="#">FDStepConditionEngine</a> (Finite-differences pricing engine for American-style vanilla options ) . . . . .	598
<a href="#">FIMCurrency</a> (Finnish markka ) . . . . .	599
<a href="#">FiniteDifferenceModel</a> (Generic finite difference model ) . . . . .	600
<a href="#">Finland</a> (Finnish calendar ) . . . . .	601
<a href="#">FixedCouponBond</a> (Fixed-coupon bond ) . . . . .	602
<a href="#">FixedCouponBondForward</a> ( <a href="#">Forward</a> contract on a fixed-coupon bond ) . . . . .	604
<a href="#">FixedCouponBondHelper</a> (Fixed-coupon bond helper ) . . . . .	607
<a href="#">FixedDividend</a> (Predetermined cash flow ) . . . . .	609
<a href="#">FixedRateCoupon</a> (Coupon paying a fixed interest rate ) . . . . .	611
<a href="#">FlatForward</a> (Flat interest-rate curve ) . . . . .	613
<a href="#">FloatingRateBond</a> (Floating-rate bond ) . . . . .	615
<a href="#">FloatingRateCoupon</a> (Coupon paying a variable index-based rate ) . . . . .	617
<a href="#">FloatingTypePayoff</a> ( <a href="#">Payoff</a> based on a floating strike ) . . . . .	620
<a href="#">Floor</a> (Concrete floor class ) . . . . .	621
<a href="#">FloorTruncation</a> ( <a href="#">Floor</a> truncation ) . . . . .	622
<a href="#">Forward</a> (Abstract base forward class ) . . . . .	623
<a href="#">ForwardEngine</a> ( <a href="#">Forward</a> engine base class ) . . . . .	626
<a href="#">ForwardFlat</a> (Forward-flat interpolation factory and traits ) . . . . .	627

<a href="#">ForwardFlatInterpolation</a> (Forward-flat interpolation between discrete points ) . . . . .	628
<a href="#">ForwardMeasureProcess</a> (Forward-measure stochastic process ) . . . . .	629
<a href="#">ForwardMeasureProcess1D</a> (Forward-measure 1-D stochastic process ) . . . . .	630
<a href="#">ForwardOptionArguments</a> (Arguments for forward (strike-resetting) option calculation ) . . . . .	631
<a href="#">ForwardPerformanceEngine</a> ( <a href="#">Forward</a> performance engine ) . . . . .	632
<a href="#">ForwardRate</a> (Forward-curve traits ) . . . . .	633
<a href="#">ForwardRateAgreement</a> ( <a href="#">Forward</a> rate agreement (FRA) class ) . . . . .	634
<a href="#">ForwardRateIpcEvolver</a> (Iterative Predictor-Corrector ) . . . . .	637
<a href="#">ForwardRatePcEvolver</a> (Predictor-Corrector ) . . . . .	638
<a href="#">ForwardRateStructure</a> ( <a href="#">Forward</a> rate term structure ) . . . . .	639
<a href="#">ForwardSpreadedTermStructure</a> (Term structure with added spread on the instantaneous forward rate ) . . . . .	641
<a href="#">ForwardTypePayoff</a> (Class for forward type payoffs ) . . . . .	643
<a href="#">ForwardVanillaOption</a> ( <a href="#">Forward</a> version of a vanilla option ) . . . . .	644
<a href="#">FractionalDividend</a> (Predetermined cash flow ) . . . . .	646
<a href="#">FraRateHelper</a> (Rate helper for bootstrapping over FRA rates ) . . . . .	648
<a href="#">FRFCurrency</a> (French franc ) . . . . .	649
<a href="#">FuturesRateHelper</a> (Rate helper for bootstrapping over interest-rate futures prices ) . . . . .	650
<a href="#">G2</a> (Two-additive-factor gaussian model class ) . . . . .	651
<a href="#">G2::FittingParameter</a> (Analytical term-structure fitting parameter $\varphi(t)$ ) . . . . .	653
<a href="#">G2ForwardProcess</a> ( <a href="#">Forward</a> G2 stochastic process ) . . . . .	654
<a href="#">G2Process</a> (G2 stochastic process ) . . . . .	656
<a href="#">G2SwaptionEngine</a> (Swaption priced by means of the Black formula ) . . . . .	658
<a href="#">GammaFunction</a> (Gamma function class ) . . . . .	659
<a href="#">GapPayoff</a> (Binary gap payoff ) . . . . .	660
<a href="#">Garch11</a> (GARCH volatility model ) . . . . .	661
<a href="#">GarmanKlassAbstract</a> . . . . .	662
<a href="#">GarmanKlassOpenClose</a> . . . . .	663
<a href="#">GarmanKohlhagenProcess</a> (Garman-Kohlhagen (1983) stochastic process ) . . . . .	664
<a href="#">GaussChebyshev2thIntegration</a> (Gauss-Chebyshev integration second kind ) . . . . .	665
<a href="#">GaussChebyshevIntegration</a> (Gauss-Chebyshev integration ) . . . . .	666
<a href="#">GaussGegenbauerIntegration</a> (Gauss-Gegenbauer integration ) . . . . .	667
<a href="#">GaussHermiteIntegration</a> (Generalized Gauss-Hermite integration ) . . . . .	668
<a href="#">GaussHermitePolynomial</a> (Gauss-Hermite polynomial ) . . . . .	669
<a href="#">GaussHyperbolicIntegration</a> (Gauss-Hyperbolic integration ) . . . . .	670
<a href="#">GaussHyperbolicPolynomial</a> (Gauss hyperbolic polynomial ) . . . . .	671
<a href="#">GaussianOrthogonalPolynomial</a> (Orthogonal polynomial for Gaussian quadratures ) . . . . .	672
<a href="#">GaussianQuadrature</a> (Integral of a 1-dimensional function using the Gauss quadratures method ) . . . . .	673
<a href="#">GaussJacobiIntegration</a> (Gauss-Jacobi integration ) . . . . .	674
<a href="#">GaussJacobiPolynomial</a> (Gauss-Jacobi polynomial ) . . . . .	675
<a href="#">GaussLaguerreIntegration</a> (Generalized Gauss-Laguerre integration ) . . . . .	676
<a href="#">GaussLaguerrePolynomial</a> (Gauss-Laguerre polynomial ) . . . . .	677
<a href="#">GaussLegendreIntegration</a> (Gauss-Legendre integration ) . . . . .	678
<a href="#">GBPCurrency</a> (British pound sterling ) . . . . .	679
<a href="#">GBPLibor</a> (GBP LIBOR rate ) . . . . .	680
<a href="#">GeneralizedBlackScholesProcess</a> (Generalized Black-Scholes stochastic process ) . . . . .	681
<a href="#">GeneralStatistics</a> (Statistics tool ) . . . . .	683
<a href="#">GenericEngine</a> (Template base class for option pricing engines ) . . . . .	686
<a href="#">GenericGaussianStatistics</a> (Statistics tool for gaussian-assumption risk measures ) . . . . .	688
<a href="#">GenericModelEngine</a> (Base class for some pricing engine on a particular model ) . . . . .	691
<a href="#">GenericRiskStatistics</a> (Empirical-distribution risk measures ) . . . . .	692
<a href="#">GenericSequenceStatistics</a> (Statistics analysis of N-dimensional (sequence) data ) . . . . .	695
<a href="#">GeometricBrownianMotionProcess</a> (Geometric brownian-motion process ) . . . . .	697



Germany (German calendars ) . . . . .	698
GRDCurrency (Greek drachma ) . . . . .	701
Greeks (Additional option results ) . . . . .	702
HaltonRsg (Halton low-discrepancy sequence generator ) . . . . .	703
Handle (Globally accessible relinkable pointer ) . . . . .	704
HestonModel (Heston model for the stochastic volatility of an asset ) . . . . .	706
HestonModelHelper (Calibration helper for Heston model ) . . . . .	707
HestonProcess (Square-root stochastic-volatility Heston process ) . . . . .	708
HKDCurrency (Honk Kong dollar ) . . . . .	710
HongKong (Hong Kong calendars ) . . . . .	711
HUFCurrency (Hungarian forint ) . . . . .	713
HullWhite (Single-factor Hull-White (extended Vasicek) model class ) . . . . .	714
HullWhite::Dynamics (Short-rate dynamics in the Hull-White model ) . . . . .	716
HullWhite::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$ ) . . . . .	717
HullWhiteForwardProcess (Forward Hull-White stochastic process ) . . . . .	718
HullWhiteProcess (Hull-White stochastic process ) . . . . .	720
Hungary (Hungarian calendar ) . . . . .	722
Iceland (Icelandic calendars ) . . . . .	723
IEPCurrency (Irish punt ) . . . . .	725
ILSCurrency (Israeli shekel ) . . . . .	726
IMM (Main cycle of the International Money Market (a.k.a. IMM) Months ) . . . . .	727
ImplicitEuler (Backward Euler scheme for finite difference methods ) . . . . .	728
ImpliedTermStructure (Implied term structure at a given date in the future ) . . . . .	729
ImpliedVolTermStructure (Implied vol term structure at a given date in the future ) . . . . .	731
InArrearIndexedCoupon (In-arrear floating-rate coupon ) . . . . .	733
IncrementalStatistics (Statistics tool based on incremental accumulation ) . . . . .	735
Index (Purely virtual base class for indexes ) . . . . .	738
IndexManager (Global repository for past index fixings ) . . . . .	740
India (Indian calendars ) . . . . .	741
Indonesia (Indonesian calendars ) . . . . .	743
INRCurrency (Indian rupee ) . . . . .	745
Instrument (Abstract instrument class ) . . . . .	746
IntegralEngine . . . . .	749
InterestRate (Concrete interest rate class ) . . . . .	750
InterestRateIndex (Base class for interest rate indexes ) . . . . .	753
InterpolatedDiscountCurve (Term structure based on interpolation of discount factors ) . . . . .	755
InterpolatedForwardCurve (Term structure based on interpolation of forward rates ) . . . . .	757
InterpolatedZeroCurve (Term structure based on interpolation of zero yields ) . . . . .	759
Interpolation (Base class for 1-D interpolations ) . . . . .	761
Interpolation2D (Base class for 2-D interpolations ) . . . . .	762
Interpolation2D::templateImpl (Basic template implementation ) . . . . .	764
Interpolation2DImpl (Abstract base class for 2-D interpolation implementations ) . . . . .	765
Interpolation::templateImpl (Basic template implementation ) . . . . .	766
InterpolationImpl (Abstract base class for interpolation implementations ) . . . . .	767
IntervalPrice (Interval price ) . . . . .	768
InverseCumulativeNormal (Inverse cumulative normal distribution function ) . . . . .	769
InverseCumulativePoisson (Inverse cumulative Poisson distribution function ) . . . . .	770
InverseCumulativeRng (Inverse cumulative random number generator ) . . . . .	771
InverseCumulativeRsg (Inverse cumulative random sequence generator ) . . . . .	772
IQDCurrency (Iraqi dinar ) . . . . .	773
IRRCurrency (Iranian rial ) . . . . .	774
ISKCurrency (Iceland krona ) . . . . .	775
Italy (Italian calendars ) . . . . .	776
ITLCurrency (Italian lira ) . . . . .	778

<a href="#">JamshidianSwaptionEngine</a> (Jamshidian swaption engine ) . . . . .	779
<a href="#">Japan</a> (Japanese calendar ) . . . . .	780
<a href="#">JarrowRudd</a> (Jarrow-Rudd (multiplicative) equal probabilities binomial tree ) . . . . .	782
<a href="#">Jibar</a> (JIBAR rate ) . . . . .	783
<a href="#">JointCalendar</a> (Joint calendar ) . . . . .	784
<a href="#">JPYCurrency</a> (Japanese yen ) . . . . .	785
<a href="#">JPYLibor</a> (JPY LIBOR rate ) . . . . .	786
<a href="#">JumpDiffusionEngine</a> (Jump-diffusion engine for vanilla options ) . . . . .	787
<a href="#">JuQuadraticApproximationEngine</a> . . . . .	788
<a href="#">KnuthUniformRng</a> (Uniform random number generator ) . . . . .	789
<a href="#">KronrodIntegral</a> (Integral of a 1-dimensional function using the Gauss-Kronrod method ) . . . . .	790
<a href="#">KRWCurrency</a> (South-Korean won ) . . . . .	791
<a href="#">KWDCurrency</a> (Kuwaiti dinar ) . . . . .	792
<a href="#">Lattice</a> (Lattice-method base class ) . . . . .	793
<a href="#">Lattice1D</a> (One-dimensional lattice ) . . . . .	795
<a href="#">Lattice2D</a> (Two-dimensional lattice ) . . . . .	796
<a href="#">LatticeShortRateModelEngine</a> (Engine for a short-rate model specialized on a lattice ) . . . . .	797
<a href="#">LazyObject</a> (Framework for calculation on demand and result caching ) . . . . .	798
<a href="#">LeastSquareFunction</a> (Cost function for least-square problems ) . . . . .	800
<a href="#">LeastSquareProblem</a> (Base class for least square problem ) . . . . .	801
<a href="#">LecuyerUniformRng</a> (Uniform random number generator ) . . . . .	802
<a href="#">LeisenReimer</a> (Leisen & Reimer tree: multiplicative approach ) . . . . .	803
<a href="#">LevenbergMarquardt</a> (Levenberg-Marquardt optimization method ) . . . . .	804
<a href="#">LexicographicalView</a> (Lexicographical 2-D view of a contiguous set of data ) . . . . .	805
<a href="#">LfmCovarianceParameterization</a> ( <a href="#">Libor</a> market model parameterization ) . . . . .	807
<a href="#">LfmCovarianceProxy</a> (Proxy for a libor forward model covariance parameterization ) . . . . .	808
<a href="#">LfmHullWhiteParameterization</a> ( <a href="#">Libor</a> market model parameterization based on Hull White paper ) . . . . .	809
<a href="#">LfmSwaptionEngine</a> ( <a href="#">Libor</a> forward model swaption engine based on black formula ) . . . . .	810
<a href="#">Libor</a> (Base class for BBA LIBOR indexes ) . . . . .	811
<a href="#">LiborForwardModel</a> ( <a href="#">Libor Forward Model</a> ) . . . . .	812
<a href="#">LiborForwardModelProcess</a> (Libor-forward-model process ) . . . . .	814
<a href="#">Linear</a> ( <a href="#">Linear</a> interpolation factory and traits ) . . . . .	816
<a href="#">LinearInterpolation</a> (Linear interpolation between discrete points ) . . . . .	817
<a href="#">LinearLeastSquaresRegression</a> (General linear least squares regression ) . . . . .	818
<a href="#">LineSearch</a> (Base class for line search ) . . . . .	819
<a href="#">Link</a> (Relinkable access to a shared pointer ) . . . . .	821
<a href="#">LmConstWrapperVolatilityModel</a> (Caplet const volatility model ) . . . . .	823
<a href="#">LmCorrelationModel</a> ( <a href="#">Libor</a> forward correlation model ) . . . . .	824
<a href="#">LmExponentialCorrelationModel</a> (Exponential correlation model ) . . . . .	825
<a href="#">LmExtLinearExponentialVolModel</a> (Extended linear exponential volatility model ) . . . . .	826
<a href="#">LmLinearExponentialCorrelationModel</a> ( <a href="#">Linear</a> exponential correlation model ) . . . . .	827
<a href="#">LmLinearExponentialVolatilityModel</a> ( <a href="#">Linear</a> exponential volatility model ) . . . . .	828
<a href="#">LmVolatilityModel</a> (Caplet volatility model ) . . . . .	829
<a href="#">LocalConstantVol</a> (Constant local volatility, no time-strike dependence ) . . . . .	830
<a href="#">LocalVolatilityEstimator</a> . . . . .	832
<a href="#">LocalVolCurve</a> (Local volatility curve derived from a Black curve ) . . . . .	833
<a href="#">LocalVolSurface</a> (Local volatility surface derived from a Black vol surface ) . . . . .	835
<a href="#">LocalVolTermStructure</a> (Local-volatility term structure ) . . . . .	837
<a href="#">LogLinear</a> (Log-linear interpolation factory and traits ) . . . . .	839
<a href="#">LogLinearInterpolation</a> . . . . .	840
<a href="#">LongstaffSchwartzPathPricer</a> (Longstaff-Schwarz path pricer for early exercise options ) . . . . .	841
<a href="#">LTLCurrency</a> (Lithuanian litas ) . . . . .	842
<a href="#">LUFCurrency</a> (Luxembourg franc ) . . . . .	843

<a href="#">LVLCurrency</a> (Latvian lat) . . . . .	844
<a href="#">MakeMCAmericanEngine</a> (Monte Carlo American engine factory) . . . . .	845
<a href="#">MakeMCDigitalEngine</a> (Monte Carlo digital engine factory) . . . . .	846
<a href="#">MakeMCEuropeanEngine</a> (Monte Carlo European engine factory) . . . . .	847
<a href="#">MakeMCEuropeanHestonEngine</a> (Monte Carlo Heston European engine factory) . . . . .	848
<a href="#">MakeMCHullWhiteCapFloorEngine</a> (Monte Carlo Hull-White cap-floor engine factory) . . . . .	849
<a href="#">MakeMCVarianceSwapEngine</a> (Monte Carlo variance-swap engine factory) . . . . .	850
<a href="#">MakeSchedule</a> (Helper class) . . . . .	851
<a href="#">MakeVanillaSwap</a> (Helper class) . . . . .	852
<a href="#">MarketModelComposite</a> (Composition of two or more market-model products) . . . . .	853
<a href="#">MarketModelEvolver</a> . . . . .	855
<a href="#">MarketModelMultiProduct</a> . . . . .	856
<a href="#">Matrix</a> (Matrix used in linear algebra) . . . . .	857
<a href="#">MCAmericanBasketEngine</a> (Least-square Monte Carlo engine) . . . . .	861
<a href="#">MCAmericanEngine</a> (American Monte Carlo engine) . . . . .	862
<a href="#">MCBarrierEngine</a> (Pricing engine for barrier options using Monte Carlo simulation) . . . . .	863
<a href="#">MCBasketEngine</a> (Pricing engine for basket options using Monte Carlo simulation) . . . . .	865
<a href="#">McCliquetOption</a> (Simple example of Monte Carlo pricer) . . . . .	867
<a href="#">MCDigitalEngine</a> (Pricing engine for digital options using Monte Carlo simulation) . . . . .	868
<a href="#">MCDiscreteArithmeticAPEngine</a> (Monte Carlo pricing engine for discrete arithmetic average price Asian) . . . . .	869
<a href="#">McDiscreteArithmeticASO</a> (Example of Monte Carlo pricer using a control variate) . . . . .	870
<a href="#">MCDiscreteAveragingAsianEngine</a> (Pricing engine for discrete average Asians using Monte Carlo simulation) . . . . .	871
<a href="#">MCDiscreteGeometricAPEngine</a> (Monte Carlo pricing engine for discrete geometric average price Asian) . . . . .	873
<a href="#">MCEuropeanEngine</a> (European option pricing engine using Monte Carlo simulation) . . . . .	874
<a href="#">MCEuropeanHestonEngine</a> (Monte Carlo Heston-model engine for European options) . . . . .	875
<a href="#">McEverest</a> (Everest-type option pricer) . . . . .	876
<a href="#">McHimalaya</a> (Himalayan-type option pricer) . . . . .	877
<a href="#">MCHullWhiteCapFloorEngine</a> (Monte Carlo Hull-White engine for cap/floors) . . . . .	878
<a href="#">MCLongstaffSchwartzEngine</a> (Longstaff Schwarz Monte Carlo engine for early exercise options) . . . . .	879
<a href="#">McMaxBasket</a> (Simple example of multi-factor Monte Carlo pricer) . . . . .	881
<a href="#">McPagoda</a> (Roofed Asian option) . . . . .	882
<a href="#">McPerformanceOption</a> (Performance option computed using Monte Carlo simulation) . . . . .	883
<a href="#">McPricer</a> (Base class for Monte Carlo pricers) . . . . .	884
<a href="#">MCSimulation</a> (Base class for Monte Carlo engines) . . . . .	885
<a href="#">MCVanillaEngine</a> (Pricing engine for vanilla options using Monte Carlo simulation) . . . . .	887
<a href="#">MCVarianceSwapEngine</a> (Variance-swap pricing engine using Monte Carlo simulation) . . . . .	888
<a href="#">MersenneTwisterUniformRng</a> (Uniform random number generator) . . . . .	890
<a href="#">Merton76Process</a> (Merton-76 jump-diffusion process) . . . . .	891
<a href="#">Mexico</a> (Mexican calendars) . . . . .	893
<a href="#">MixedScheme</a> (Mixed (explicit/implicit) scheme for finite difference methods) . . . . .	895
<a href="#">Money</a> (Amount of cash) . . . . .	897
<a href="#">MonotonicCubicSpline</a> (Cubic spline with monotonicity constraint) . . . . .	899
<a href="#">MonteCarloModel</a> (General purpose Monte Carlo model for path samples) . . . . .	900
<a href="#">MoreGreeks</a> (More additional option results) . . . . .	901
<a href="#">MoroInverseCumulativeNormal</a> (Moro Inverse cumulative normal distribution class) . . . . .	902
<a href="#">MTBrownianGenerator</a> . . . . .	903
<a href="#">MTLCurrency</a> (Maltese lira) . . . . .	904
<a href="#">MultiAssetOption</a> (Base class for options on multiple assets) . . . . .	905
<a href="#">MultiAssetOption::arguments</a> (Arguments for multi-asset option calculation) . . . . .	907
<a href="#">MultiAssetOption::results</a> (Results from multi-asset option calculation) . . . . .	908



MultiCubicSpline	909
MultiPath (Correlated multiple asset paths)	910
MultiPathGenerator (Generates a multipath from a random number generator)	911
MultiProductComposite (Composition of one or more market-model products)	912
MultiProductMultiStep	913
MultiProductOneStep	914
MultiVariate (Default Monte Carlo traits for multi-variate models)	915
MXNCurrency (Mexican peso)	916
NaturalCubicSpline (Cubic spline with null second derivative at end points)	917
NaturalMonotonicCubicSpline (Natural cubic spline with monotonicity constraint)	918
NeumannBC (Neumann boundary condition (i.e., constant derivative))	919
Newton (Newton 1-D solver)	920
NewtonSafe (Safe Newton 1-D solver)	921
NewZealand (New Zealand calendar)	922
NLGCurrency (Dutch guilder)	923
NoConstraint (No constraint)	924
NOKCurrency (Norwegian krone)	925
NonLinearLeastSquare (Non-linear least-square method)	926
NormalDistribution (Normal distribution function)	927
Norway (Norwegian calendar)	928
NPRCurrency (Nepal rupee)	929
Null (Template class providing a null value for a given type)	930
NullCalendar (Calendar for reproducing theoretical calculations)	931
NullCondition (Null step condition)	932
NullParameter (Parameter which is always zero $a(t) = 0$ )	933
NumericalMethod (Numerical method (tree, finite-differences) base class)	934
NZDCurrency (New Zealand dollar)	936
NZDLibor (NZD LIBOR rate)	937
Observable (Object that notifies its changes to a set of observables)	938
ObservableValue (Observable and assignable proxy to concrete value)	940
Observer (Object that gets notified when a given observable changes)	941
OneAssetOption (Base class for options on a single asset)	943
OneAssetOption::arguments (Arguments for single-asset option calculation)	946
OneAssetOption::results (Results from single-asset option calculation)	947
OneAssetStrikedOption (Base class for options on a single asset with striked payoff)	948
OneDayCounter (1/1 day count convention)	950
OneFactorAffineModel (Single-factor affine base class)	951
OneFactorModel (Single-factor short-rate model abstract class)	952
OneFactorModel::ShortRateDynamics (Base class describing the short-rate dynamics)	953
OneFactorModel::ShortRateTree (Recombining trinomial tree discretizing the state variable)	954
OperatorFactory (Black-Scholes-Merton differential operator)	955
OptimizationMethod (Abstract class for constrained optimization method)	956
Option (Base option class)	958
Option::arguments	959
OrnsteinUhlenbeckProcess (Ornstein-Uhlenbeck process class)	960
Parameter (Base class for model arguments)	962
ParameterImpl (Base class for model parameter implementation)	963
ParCoupon (coupon at par on a term structure)	964
Path	965
PathGenerator (Generates random paths using a sequence generator)	966
PathPricer (Base class for path pricers)	967
Payoff (Base class for option payoffs)	968
PercentageStrikePayoff (Payoff with strike expressed as percentage)	969

<a href="#">Period</a> (Time period described by a number of a given time unit ) . . . . .	970
<a href="#">PiecewiseConstantParameter</a> (Piecewise-constant parameter ) . . . . .	971
<a href="#">PiecewiseYieldCurve</a> (Piecewise yield term structure ) . . . . .	972
<a href="#">PiecewiseZeroSpreadedTermStructure</a> (Term structure with an added vector of spreads on the zero-yield rate ) . . . . .	974
<a href="#">PKRCurrency</a> (Pakistani rupee ) . . . . .	976
<a href="#">PlainVanillaPayoff</a> (Plain-vanilla payoff ) . . . . .	977
<a href="#">PLNCurrency</a> (Polish zloty ) . . . . .	978
<a href="#">PoissonDistribution</a> (Normal distribution function ) . . . . .	979
<a href="#">Poland</a> (Polish calendar ) . . . . .	980
<a href="#">PositiveConstraint</a> (Constraint imposing positivity to all arguments ) . . . . .	981
<a href="#">PriceCurve</a> (Additional pricing results ) . . . . .	982
<a href="#">PricingEngine</a> (Interface for pricing engines ) . . . . .	983
<a href="#">PrimeNumbers</a> (Prime numbers calculator ) . . . . .	984
<a href="#">Problem</a> (Constrained optimization problem ) . . . . .	985
<a href="#">PTECurrency</a> (Portuguese escudo ) . . . . .	987
<a href="#">QuantoEngine</a> (Quanto engine base class ) . . . . .	988
<a href="#">QuantoForwardVanillaOption</a> (Quanto version of a forward vanilla option ) . . . . .	990
<a href="#">QuantoOptionArguments</a> (Arguments for quanto option calculation ) . . . . .	992
<a href="#">QuantoOptionResults</a> (Results from quanto option calculation ) . . . . .	993
<a href="#">QuantoTermStructure</a> (Quanto term structure ) . . . . .	994
<a href="#">QuantoVanillaOption</a> (Quanto version of a vanilla option ) . . . . .	996
<a href="#">Quote</a> (Purely virtual base class for market observables ) . . . . .	998
<a href="#">RandomizedLDS</a> (Randomized (random shift) low-discrepancy sequence ) . . . . .	999
<a href="#">RandomSequenceGenerator</a> (Random sequence generator based on a pseudo-random number generator ) . . . . .	1001
<a href="#">RateHelper</a> (Base helper class for yield-curve bootstrapping ) . . . . .	1002
<a href="#">RelativeDateRateHelper</a> (Rate helper with date schedule relative to the global evaluation date ) . . . . .	1004
<a href="#">ReplicatingVarianceSwapEngine</a> (Variance-swap pricing engine using replicating cost, ) . . . . .	1005
<a href="#">Results</a> (Base class for generic result groups ) . . . . .	1006
<a href="#">Ridder</a> (Ridder 1-D solver ) . . . . .	1007
<a href="#">ROLCurrency</a> (Romanian leu ) . . . . .	1008
<a href="#">RONCurrency</a> (Romanian new leu ) . . . . .	1009
<a href="#">Rounding</a> (Basic rounding class ) . . . . .	1010
<a href="#">SABRInterpolation</a> (SABR smile interpolation between discrete volatility points ) . . . . .	1012
<a href="#">SalvagingAlgorithm</a> (Algorithm used for matricial pseudo square root ) . . . . .	1013
<a href="#">Sample</a> (Weighted sample ) . . . . .	1014
<a href="#">SampledCurve</a> (This class contains a sampled curve ) . . . . .	1015
<a href="#">SARCurrency</a> (Saudi riyal ) . . . . .	1017
<a href="#">SaudiArabia</a> (Saudi Arabian calendar ) . . . . .	1018
<a href="#">Schedule</a> (Payment schedule ) . . . . .	1019
<a href="#">Secant</a> (Secant 1-D solver ) . . . . .	1021
<a href="#">SeedGenerator</a> (Random seed generator ) . . . . .	1022
<a href="#">SegmentIntegral</a> (Integral of a one-dimensional function ) . . . . .	1023
<a href="#">SEKCurrency</a> (Swedish krona ) . . . . .	1024
<a href="#">Settings</a> (Global repository for run-time library settings ) . . . . .	1025
<a href="#">Settlement</a> ( <a href="#">Settlement</a> information ) . . . . .	1027
<a href="#">SGDCurrency</a> ( <a href="#">Singapore</a> dollar ) . . . . .	1028
<a href="#">Short</a> (Short indexed coupon ) . . . . .	1029
<a href="#">Short&lt; ParCoupon &gt;</a> (Short coupon at par on a term structure ) . . . . .	1030
<a href="#">ShortRateModel</a> (Abstract short-rate model class ) . . . . .	1031
<a href="#">ShoutCondition</a> (Shout option condition ) . . . . .	1032
<a href="#">SimpleCashFlow</a> (Predetermined cash flow ) . . . . .	1033

<a href="#">SimpleDayCounter</a> (Simple day counter for reproducing theoretical calculations ) . . . .	1034
<a href="#">SimpleLocalEstimator</a> . . . . .	1035
<a href="#">SimpleQuote</a> (Market element returning a stored value ) . . . . .	1036
<a href="#">Simplex</a> (Multi-dimensional simplex class ) . . . . .	1037
<a href="#">SimpsonIntegral</a> (Integral of a one-dimensional function ) . . . . .	1038
<a href="#">Singapore</a> (Singapore calendars ) . . . . .	1039
<a href="#">SingleAssetOption</a> (Black-Scholes-Merton option ) . . . . .	1041
<a href="#">SingleProductComposite</a> (Composition of one or more market-model products ) . . . .	1043
<a href="#">Singleton</a> (Basic support for the singleton pattern ) . . . . .	1044
<a href="#">SingleVariate</a> (Default Monte Carlo traits for single-variate models ) . . . . .	1045
<a href="#">SITCurrency</a> (Slovenian tolar ) . . . . .	1046
<a href="#">SKKCurrency</a> (Slovak koruna ) . . . . .	1047
<a href="#">Slovakia</a> (Slovak calendars ) . . . . .	1048
<a href="#">SmileSection</a> ( <a href="#">Swaption</a> volatility smile section ) . . . . .	1050
<a href="#">SobolRsg</a> (Sobol low-discrepancy sequence generator ) . . . . .	1051
<a href="#">SoftCallability</a> ( <a href="#">Callability</a> leaving to the holder the possibility to convert ) . . . . .	1053
<a href="#">Solver1D</a> (Base class for 1-D solvers ) . . . . .	1054
<a href="#">SouthAfrica</a> (South-African calendar ) . . . . .	1056
<a href="#">SouthKorea</a> (South Korean calendars ) . . . . .	1057
<a href="#">SquareRootProcess</a> (Square-root process class ) . . . . .	1059
<a href="#">StatsHolder</a> (Helper class for precomputed distributions ) . . . . .	1060
<a href="#">SteepestDescent</a> (Multi-dimensional steepest-descent class ) . . . . .	1061
<a href="#">step_iterator</a> (Iterator advancing in constant steps ) . . . . .	1062
<a href="#">StepCondition</a> (Condition to be applied at every time step ) . . . . .	1063
<a href="#">StepConditionSet</a> (Parallel evolver for multiple arrays ) . . . . .	1064
<a href="#">StochasticProcess</a> (Multi-dimensional stochastic process class ) . . . . .	1065
<a href="#">StochasticProcess1D</a> (1-dimensional stochastic process ) . . . . .	1068
<a href="#">StochasticProcess1D::discretization</a> (Discretization of a 1-D stochastic process ) . . . .	1070
<a href="#">StochasticProcess::discretization</a> (Discretization of a stochastic process over a given time interval ) . . . . .	1071
<a href="#">StochasticProcessArray</a> (Array of correlated 1-D stochastic processes ) . . . . .	1072
<a href="#">Stock</a> (Simple stock class ) . . . . .	1074
<a href="#">StrikedTypePayoff</a> (Intermediate class for payoffs based on a fixed strike ) . . . . .	1075
<a href="#">StulzEngine</a> (Pricing engine for 2D European Baskets ) . . . . .	1076
<a href="#">SuperSharePayoff</a> (Binary supershare payoff ) . . . . .	1077
<a href="#">SVD</a> (Singular value decomposition ) . . . . .	1078
<a href="#">Swap</a> (Interest rate swap ) . . . . .	1079
<a href="#">SwapIndex</a> (Base class for swap-rate indexes ) . . . . .	1081
<a href="#">SwapRateHelper</a> (Rate helper for bootstrapping over swap rates ) . . . . .	1083
<a href="#">Swaption</a> (Swaption class ) . . . . .	1085
<a href="#">Swaption::arguments</a> (Arguments for swaption calculation ) . . . . .	1087
<a href="#">Swaption::engine</a> (Base class for swaption engines ) . . . . .	1088
<a href="#">Swaption::results</a> (Results from swaption calculation ) . . . . .	1089
<a href="#">SwaptionConstantVolatility</a> (Constant swaption volatility, no time-strike dependence ) .	1090
<a href="#">SwaptionHelper</a> (Calibration helper for ATM swaption ) . . . . .	1092
<a href="#">SwaptionVolatilityCube</a> . . . . .	1093
<a href="#">SwaptionVolatilityCubeBySabr</a> . . . . .	1095
<a href="#">SwaptionVolatilityMatrix</a> (At-the-money swaption-volatility matrix ) . . . . .	1097
<a href="#">SwaptionVolatilityStructure</a> (Swaption-volatility structure ) . . . . .	1099
<a href="#">Sweden</a> (Swedish calendar ) . . . . .	1102
<a href="#">Switzerland</a> (Swiss calendar ) . . . . .	1103
<a href="#">SymmetricSchurDecomposition</a> (Symmetric threshold Jacobi algorithm ) . . . . .	1104
<a href="#">TabulatedGaussLegendre</a> (Tabulated Gauss-Legendre quadratures ) . . . . .	1105
<a href="#">Taiwan</a> (Taiwanese calendars ) . . . . .	1106

TARGET (TARGET calendar ) . . . . .	1108
TermStructure (Basic term-structure functionality ) . . . . .	1109
TermStructureConsistentModel (Term-structure consistent model class ) . . . . .	1111
TermStructureFittingParameter (Deterministic time-dependent parameter used for yield-curve fitting ) . . . . .	1112
THBCurrency (Thai baht ) . . . . .	1113
Thirty360 (30/360 day count convention ) . . . . .	1114
Tian (Tian tree: third moment matching, multiplicative approach ) . . . . .	1115
Tibor (JPY TIBOR index ) . . . . .	1116
TimeBasket (Distribution over a number of dates ) . . . . .	1117
TimeGrid (Time grid class ) . . . . .	1118
TimeSeries (Container for historical data ) . . . . .	1120
TqrEigenDecomposition (Tridiag. QR eigen decomposition with explicite shift aka Wilkinson ) . . . . .	1122
TransformedGrid (Transformed grid ) . . . . .	1123
TrapezoidIntegral (Integral of a one-dimensional function ) . . . . .	1124
Tree (Tree approximating a single-factor diffusion ) . . . . .	1126
TreeCapFloorEngine (Numerical lattice engine for cap/floors ) . . . . .	1127
TreeSwaptionEngine (Numerical lattice engine for swaptions ) . . . . .	1128
TreeVanillaSwapEngine (Numerical lattice engine for simple swaps ) . . . . .	1129
TridiagonalOperator (Base implementation for tridiagonal operator ) . . . . .	1130
TridiagonalOperator::TimeSetter (Encapsulation of time-setting logic ) . . . . .	1132
Trigeorgis (Trigeorgis (additive equal jumps) binomial tree ) . . . . .	1133
TrinomialTree (Recombining trinomial tree class ) . . . . .	1134
TRLCurrency (Turkish lira ) . . . . .	1135
TRLibor (TRY LIBOR rate ) . . . . .	1136
TRYCurrency (New Turkish lira ) . . . . .	1137
TsiveriotisFernandesLattice (Binomial lattice approximating the Tsiveriotis-Fernandes model ) . . . . .	1138
TTDCurrency (Trinidad & Tobago dollar ) . . . . .	1140
Turkey (Turkish calendar ) . . . . .	1141
TWDCurrency (Taiwan dollar ) . . . . .	1142
TwoFactorModel (Abstract base-class for two-factor models ) . . . . .	1143
TwoFactorModel::ShortRateDynamics (Class describing the dynamics of the two state variables ) . . . . .	1144
TwoFactorModel::ShortRateTree (Recombining two-dimensional tree discretizing the state variable ) . . . . .	1145
TypePayoff (Intermediate class for call/put/straddle payoffs ) . . . . .	1146
Ukraine (Ukrainian calendars ) . . . . .	1147
UnitedKingdom (United Kingdom calendars ) . . . . .	1149
UnitedStates (United States calendars ) . . . . .	1151
UpFrontIndexedCoupon (up front indexed coupon class ) . . . . .	1154
UpRounding (Up-rounding ) . . . . .	1155
USDCurrency (U.S. dollar ) . . . . .	1156
USDLibor (USD LIBOR rate ) . . . . .	1157
Value (Pricing results ) . . . . .	1158
VanillaCMSCouponPricer (Pricer for vanilla CMS coupons ) . . . . .	1159
VanillaOption (Vanilla option (no discrete dividends, no barriers) on a single asset ) . .	1160
VanillaOption::engine (Vanilla option engine base class ) . . . . .	1161
VanillaSwap (Plain-vanilla swap ) . . . . .	1162
VanillaSwap::arguments (Arguments for simple swap calculation ) . . . . .	1164
VanillaSwap::results (Results from simple swap calculation ) . . . . .	1165
VarianceSwap (Variance swap ) . . . . .	1166
VarianceSwap::arguments (Arguments for forward fair-variance calculation ) . . . . .	1169

<a href="#">VarianceSwap::engine</a> (Base class for variance-swap engines ) . . . . .	1170
<a href="#">VarianceSwap::results</a> (Results from variance-swap calculation ) . . . . .	1171
<a href="#">Vasicek</a> (Vasicek model class ) . . . . .	1172
<a href="#">Vasicek::Dynamics</a> (Short-rate dynamics in the Vasicek model ) . . . . .	1174
<a href="#">VEBCurrency</a> (Venezuelan bolivar ) . . . . .	1175
<a href="#">Visitor</a> (Visitor for a specific class ) . . . . .	1176
<a href="#">Xibor</a> (Base class for LIBOR-like indexes ) . . . . .	1177
<a href="#">YieldTermStructure</a> (Interest-rate term structure ) . . . . .	1179
<a href="#">ZARCurrency</a> (South-African rand ) . . . . .	1183
<a href="#">ZeroCondition</a> (Zero exercise condition ) . . . . .	1184
<a href="#">ZeroCouponBond</a> (Zero-coupon bond ) . . . . .	1185
<a href="#">ZeroSpreadedTermStructure</a> (Term structure with an added spread on the zero yield rate )	1186
<a href="#">ZeroYield</a> (Zero-curve traits ) . . . . .	1188
<a href="#">ZeroYieldStructure</a> (Zero-yield term structure ) . . . . .	1189
<a href="#">Zibor</a> (CHF ZIBOR rate ) . . . . .	1190



## Chapter 5

# QuantLib File Index

### 5.1 QuantLib File List

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

<a href="#">ql/argsandresults.hpp</a> (Base classes for generic arguments and results ) . . . . .	1191
<a href="#">ql/calendar.hpp</a> (calendar class ) . . . . .	1193
<a href="#">ql/capvolstructures.hpp</a> (Cap/Floor volatility structures ) . . . . .	1230
<a href="#">ql/cashflow.hpp</a> (Base class for cash flows ) . . . . .	1231
<a href="#">ql/currency.hpp</a> (Known currencies ) . . . . .	1259
<a href="#">ql/date.hpp</a> (Date- and time-related classes, typedefs and enumerations ) . . . . .	1260
<a href="#">ql/daycounter.hpp</a> (Day counter class ) . . . . .	1263
<a href="#">ql/discretizedasset.hpp</a> (Discretized asset classes ) . . . . .	1271
<a href="#">ql/errors.hpp</a> (Classes and functions for error handling ) . . . . .	1272
<a href="#">ql/event.hpp</a> (Base class for events associated with a given date ) . . . . .	1275
<a href="#">ql/exchangerate.hpp</a> (Exchange rate between two currencies ) . . . . .	1276
<a href="#">ql/exercise.hpp</a> (Option exercise classes and payoff function ) . . . . .	1277
<a href="#">ql/grid.hpp</a> (Grid constructors ) . . . . .	1303
<a href="#">ql/handle.hpp</a> (Globally accessible relinkable pointer ) . . . . .	1304
<a href="#">ql/index.hpp</a> (Purely virtual base class for indexes ) . . . . .	1305
<a href="#">ql/instrument.hpp</a> (Abstract instrument class ) . . . . .	1339
<a href="#">ql/interestrates.hpp</a> (Instrument rate class ) . . . . .	1375
<a href="#">ql/money.hpp</a> (Cash amount in a given currency ) . . . . .	1440
<a href="#">ql/numericalmethod.hpp</a> (Numerical method class ) . . . . .	1455
<a href="#">ql/option.hpp</a> (Base option class ) . . . . .	1469
<a href="#">ql/payoff.hpp</a> (Option payoff classes ) . . . . .	1477
<a href="#">ql/period.hpp</a> (Period- and frequency-related classes and enumerations ) . . . . .	1478
<a href="#">ql/position.hpp</a> (Short or long position ) . . . . .	1480
<a href="#">ql/prices.hpp</a> (Price classes ) . . . . .	1491
<a href="#">ql/pricingengine.hpp</a> (Base class for pricing engines ) . . . . .	1492
<a href="#">ql/qldefines.hpp</a> (Global definitions and compiler switches ) . . . . .	1577
<a href="#">ql/quote.hpp</a> (Purely virtual base class for market observables ) . . . . .	1579
<a href="#">ql/schedule.hpp</a> (Date schedule ) . . . . .	1595
<a href="#">ql/settings.hpp</a> (Global repository for run-time library settings ) . . . . .	1596
<a href="#">ql/solver1d.hpp</a> (Abstract 1-D solver class ) . . . . .	1624
<a href="#">ql/stochasticprocess.hpp</a> (Stochastic processes ) . . . . .	1632
<a href="#">ql/swaptionvolstructure.hpp</a> (Swaption volatility structure ) . . . . .	1633
<a href="#">ql/termstructure.hpp</a> (Base class for term structures ) . . . . .	1634



ql/timegrid.hpp (Discrete time grid ) . . . . .	1654
ql/timeseries.hpp (Timeseries class ) . . . . .	1655
ql/types.hpp (Custom types ) . . . . .	1656
ql/volatilitymodel.hpp (Volatility term structures ) . . . . .	1684
ql/voltermstructure.hpp (Volatility term structures ) . . . . .	1689
ql/yieldtermstructure.hpp (Interest-rate term structure ) . . . . .	1690
ql/Calendars/argentina.hpp (Argentinian calendars ) . . . . .	1195
ql/Calendars/australia.hpp (Australian calendar ) . . . . .	1196
ql/Calendars/brazil.hpp (Brazilian calendar ) . . . . .	1197
ql/Calendars/canada.hpp (Canadian calendar ) . . . . .	1198
ql/Calendars/china.hpp (Chinese calendar ) . . . . .	1199
ql/Calendars/czechrepublic.hpp (Czech calendars ) . . . . .	1200
ql/Calendars/denmark.hpp (Danish calendar ) . . . . .	1201
ql/Calendars/finland.hpp (Finnish calendar ) . . . . .	1202
ql/Calendars/germany.hpp (German calendars ) . . . . .	1203
ql/Calendars/hongkong.hpp (Hong Kong calendars ) . . . . .	1204
ql/Calendars/hungary.hpp (Hungarian calendar ) . . . . .	1205
ql/Calendars/iceland.hpp (Icelandic calendars ) . . . . .	1206
ql/Calendars/india.hpp (Indian calendars ) . . . . .	1207
ql/Calendars/indonesia.hpp (Indonesian calendars ) . . . . .	1208
ql/Calendars/italy.hpp (Italian calendars ) . . . . .	1209
ql/Calendars/japan.hpp (Japanese calendar ) . . . . .	1210
ql/Calendars/jointcalendar.hpp (Joint calendar ) . . . . .	1211
ql/Calendars/mexico.hpp (Mexican calendars ) . . . . .	1212
ql/Calendars/newzealand.hpp (New Zealand calendar ) . . . . .	1213
ql/Calendars/norway.hpp (Norwegian calendar ) . . . . .	1214
ql/Calendars/nullcalendar.hpp (Calendar for reproducing theoretical calculations ) . . . . .	1215
ql/Calendars/poland.hpp (Polish calendar ) . . . . .	1216
ql/Calendars/saudiarabia.hpp (Saudi Arabian calendar ) . . . . .	1217
ql/Calendars/singapore.hpp (Singapore calendars ) . . . . .	1218
ql/Calendars/slovakia.hpp (Slovak calendars ) . . . . .	1219
ql/Calendars/southafrica.hpp (South-African calendar ) . . . . .	1220
ql/Calendars/southkorea.hpp (South Korean calendars ) . . . . .	1221
ql/Calendars/sweden.hpp (Swedish calendar ) . . . . .	1222
ql/Calendars/switzerland.hpp (Swiss calendar ) . . . . .	1223
ql/Calendars/taiwan.hpp (Taiwanese calendars ) . . . . .	1224
ql/Calendars/target.hpp (TARGET calendar ) . . . . .	1225
ql/Calendars/turkey.hpp (Turkish calendar ) . . . . .	1226
ql/Calendars/ukraine.hpp (Ukrainian calendars ) . . . . .	1227
ql/Calendars/unitedkingdom.hpp (UK calendars ) . . . . .	1228
ql/Calendars/unitedstates.hpp (US calendars ) . . . . .	1229
ql/CashFlows/analysis.hpp (Cash-flow analysis functions ) . . . . .	1232
ql/CashFlows/cashflowvectors.hpp (Cash flow vector builders ) . . . . .	1233
ql/CashFlows/cmscoupon.hpp (CMS coupon ) . . . . .	1234
ql/CashFlows/conundrumpricer.hpp . . . . .	1236
ql/CashFlows/coupon.hpp (Coupon accruing over a fixed period ) . . . . .	1237
ql/CashFlows/dividend.hpp (A stock dividend ) . . . . .	1238
ql/CashFlows/fixedratecoupon.hpp (Coupon paying a fixed annual rate ) . . . . .	1239
ql/CashFlows/floatingratecoupon.hpp (Coupon paying a variable index-based rate ) . . . . .	1240
ql/CashFlows/inarrearindexedcoupon.hpp (In-arrear floating-rate coupon ) . . . . .	1241
ql/CashFlows/indexedcashflowvectors.hpp (Indexed cash-flow vector builders ) . . . . .	1242
ql/CashFlows/indexedcoupon.hpp (Indexed coupon ) . . . . .	1243
ql/CashFlows/parcoupon.hpp (Coupon at par on a term structure ) . . . . .	1244
ql/CashFlows/shortfloatingcoupon.hpp (Short (or long) coupon at par on a term structure ) . . . . .	1245



ql/CashFlows/ <a href="#">shortindexedcoupon.hpp</a> (Short (or long) indexed coupon ) . . . . .	1246
ql/CashFlows/ <a href="#">simplecashflow.hpp</a> (Predetermined cash flow ) . . . . .	1247
ql/CashFlows/ <a href="#">timebasket.hpp</a> . . . . .	1248
ql/CashFlows/ <a href="#">upfrontindexedcoupon.hpp</a> (Up front indexed coupon ) . . . . .	1249
ql/Currencies/ <a href="#">africa.hpp</a> (African currencies ) . . . . .	1250
ql/Currencies/ <a href="#">america.hpp</a> (American currencies ) . . . . .	1251
ql/Currencies/ <a href="#">asia.hpp</a> (Asian currencies ) . . . . .	1252
ql/Currencies/ <a href="#">europe.hpp</a> (European currencies ) . . . . .	1254
ql/Currencies/ <a href="#">exchangeratemanager.hpp</a> (Exchange-rate repository ) . . . . .	1257
ql/Currencies/ <a href="#">oceania.hpp</a> (Oceanian currencies ) . . . . .	1258
ql/DayCounters/ <a href="#">actual360.hpp</a> (Act/360 day counter ) . . . . .	1264
ql/DayCounters/ <a href="#">actual365fixed.hpp</a> (Actual/365 (Fixed) day counter ) . . . . .	1265
ql/DayCounters/ <a href="#">actualactual.hpp</a> (Act/act day counters ) . . . . .	1266
ql/DayCounters/ <a href="#">business252.hpp</a> (Business/252 day counter ) . . . . .	1267
ql/DayCounters/ <a href="#">one.hpp</a> (1/1 day counter ) . . . . .	1268
ql/DayCounters/ <a href="#">simplifiedaycounter.hpp</a> (Simple day counter for reproducing theoretical calculations ) . . . . .	1269
ql/DayCounters/ <a href="#">thirty360.hpp</a> (30/360 day counters ) . . . . .	1270
ql/FiniteDifferences/ <a href="#">americancondition.hpp</a> (American option exercise condition ) . . . . .	1278
ql/FiniteDifferences/ <a href="#">boundarycondition.hpp</a> (Boundary conditions for differential operators ) . . . . .	1279
ql/FiniteDifferences/ <a href="#">bsmoperator.hpp</a> (Differential operator for Black-Scholes-Merton equation ) . . . . .	1280
ql/FiniteDifferences/ <a href="#">bsmtermoperator.hpp</a> (Differential operator for Black-Scholes-Merton equation ) . . . . .	1281
ql/FiniteDifferences/ <a href="#">cranknicolson.hpp</a> (Crank-Nicolson scheme for finite difference methods ) . . . . .	1282
ql/FiniteDifferences/ <a href="#">dminus.hpp</a> ( $D_-$ matricial representation ) . . . . .	1283
ql/FiniteDifferences/ <a href="#">dplus.hpp</a> ( $D_+$ matricial representation ) . . . . .	1284
ql/FiniteDifferences/ <a href="#">dplusdminus.hpp</a> ( $D_+D_-$ matricial representation ) . . . . .	1285
ql/FiniteDifferences/ <a href="#">dzero.hpp</a> ( $D_0$ matricial representation ) . . . . .	1286
ql/FiniteDifferences/ <a href="#">expliciteuler.hpp</a> (Explicit Euler scheme for finite difference methods ) . . . . .	1287
ql/FiniteDifferences/ <a href="#">fdtypedefs.hpp</a> (Default choices for template instantiations ) . . . . .	1288
ql/FiniteDifferences/ <a href="#">finitedifferencemodel.hpp</a> (Generic finite difference model ) . . . . .	1289
ql/FiniteDifferences/ <a href="#">impliciteuler.hpp</a> (Implicit Euler scheme for finite difference methods ) . . . . .	1290
ql/FiniteDifferences/ <a href="#">mixedscheme.hpp</a> (Mixed (explicit/implicit) scheme for finite difference methods ) . . . . .	1291
ql/FiniteDifferences/ <a href="#">onefactoroperator.hpp</a> (General differential operator for one-factor interest rate models ) . . . . .	1292
ql/FiniteDifferences/ <a href="#">operatorfactory.hpp</a> (Factory for finite difference operators ) . . . . .	1293
ql/FiniteDifferences/ <a href="#">operatortraits.hpp</a> (Differential operator traits ) . . . . .	1294
ql/FiniteDifferences/ <a href="#">parallelevolver.hpp</a> (Parallel evolver for multiple arrays ) . . . . .	1295
ql/FiniteDifferences/ <a href="#">pde.hpp</a> (General class for one dimensional PDE's ) . . . . .	1296
ql/FiniteDifferences/ <a href="#">pdebsm.hpp</a> (Black-Scholes-Merton PDE ) . . . . .	1297
ql/FiniteDifferences/ <a href="#">pdeshortrate.hpp</a> (Adapter to short rate ) . . . . .	1298
ql/FiniteDifferences/ <a href="#">shoutcondition.hpp</a> (Shout option exercise condition ) . . . . .	1299
ql/FiniteDifferences/ <a href="#">stepcondition.hpp</a> (Conditions to be applied at every time step ) . . . . .	1300
ql/FiniteDifferences/ <a href="#">tridiagonaloperator.hpp</a> (Tridiagonal operator ) . . . . .	1301
ql/FiniteDifferences/ <a href="#">zerocondition.hpp</a> (Zero option exercise condition ) . . . . .	1302
ql/Indexes/ <a href="#">audlibor.hpp</a> (AUD LIBOR rate ) . . . . .	1306
ql/Indexes/ <a href="#">cadlibor.hpp</a> (CAD LIBOR rate ) . . . . .	1307
ql/Indexes/ <a href="#">cdor.hpp</a> (CDOR rate ) . . . . .	1308

ql/Indexes/ <a href="#">chflibor.hpp</a> (CHF LIBOR rate ) . . . . .	1309
ql/Indexes/ <a href="#">dkklibor.hpp</a> (DKK LIBOR rate ) . . . . .	1310
ql/Indexes/ <a href="#">euribor.hpp</a> (Euribor index ) . . . . .	1311
ql/Indexes/ <a href="#">euriborswapfixa.hpp</a> (euriborswapfixa index ) . . . . .	1314
ql/Indexes/ <a href="#">euriborswapfixifr.hpp</a> (EuriborSwapFixIFR index ) . . . . .	1316
ql/Indexes/ <a href="#">eurlibor.hpp</a> (EUR LIBOR rate ) . . . . .	1318
ql/Indexes/ <a href="#">eurliborswapfixa.hpp</a> (eurliborswapfixa index ) . . . . .	1320
ql/Indexes/ <a href="#">eurliborswapfixb.hpp</a> (eurliborswapfixb index ) . . . . .	1322
ql/Indexes/ <a href="#">eurliborswapfixifr.hpp</a> (eurliborswapfixifr index ) . . . . .	1324
ql/Indexes/ <a href="#">gbplibor.hpp</a> (GBP LIBOR rate ) . . . . .	1326
ql/Indexes/ <a href="#">indexmanager.hpp</a> (Global repository for past index fixings ) . . . . .	1327
ql/Indexes/ <a href="#">interestrateindex.hpp</a> (Base class for interest rate indexes ) . . . . .	1328
ql/Indexes/ <a href="#">jibar.hpp</a> (JIBAR rate ) . . . . .	1329
ql/Indexes/ <a href="#">jpylibor.hpp</a> (JPY LIBOR rate ) . . . . .	1330
ql/Indexes/ <a href="#">libor.hpp</a> (Base class for BBA LIBOR indexes ) . . . . .	1331
ql/Indexes/ <a href="#">nzdlibor.hpp</a> (NZD LIBOR rate ) . . . . .	1332
ql/Indexes/ <a href="#">swapindex.hpp</a> (Swap-rate indexes ) . . . . .	1333
ql/Indexes/ <a href="#">tibor.hpp</a> (JPY TIBOR rate ) . . . . .	1334
ql/Indexes/ <a href="#">trlibor.hpp</a> (TRY LIBOR rate ) . . . . .	1335
ql/Indexes/ <a href="#">usdlibor.hpp</a> (USD LIBOR rate ) . . . . .	1336
ql/Indexes/ <a href="#">xibor.hpp</a> (Base class for LIBOR-like indexes ) . . . . .	1337
ql/Indexes/ <a href="#">zibor.hpp</a> (CHF ZIBOR rate ) . . . . .	1338
ql/Instruments/ <a href="#">asianoption.hpp</a> (Asian option on a single asset ) . . . . .	1340
ql/Instruments/ <a href="#">assetswap.hpp</a> (Bullet Bond vs Libor swap ) . . . . .	1341
ql/Instruments/ <a href="#">barrieroption.hpp</a> (Barrier option on a single asset ) . . . . .	1342
ql/Instruments/ <a href="#">basketoption.hpp</a> (Basket option on a number of assets ) . . . . .	1343
ql/Instruments/ <a href="#">bond.hpp</a> (Concrete bond class ) . . . . .	1344
ql/Instruments/ <a href="#">callabilityschedule.hpp</a> (Schedule of put/call dates ) . . . . .	1345
ql/Instruments/ <a href="#">capfloor.hpp</a> (Cap and Floor class ) . . . . .	1346
ql/Instruments/ <a href="#">cliquetoption.hpp</a> (Cliquet option ) . . . . .	1348
ql/Instruments/ <a href="#">compositeinstrument.hpp</a> (Composite instrument class ) . . . . .	1349
ql/Instruments/ <a href="#">convertiblebond.hpp</a> (Convertible bond class ) . . . . .	1350
ql/Instruments/ <a href="#">dividendschedule.hpp</a> (Schedule of dividend dates ) . . . . .	1352
ql/Instruments/ <a href="#">dividendvanillaoption.hpp</a> (Vanilla option on a single asset with discrete dividends ) . . . . .	1353
ql/Instruments/ <a href="#">europeanoption.hpp</a> (European option on a single asset ) . . . . .	1354
ql/Instruments/ <a href="#">fixedcouponbond.hpp</a> (Fixed-coupon bond ) . . . . .	1355
ql/Instruments/ <a href="#">fixedcouponbondforward.hpp</a> (Forward contract on a fixed-coupon bond ) . . . . .	1356
ql/Instruments/ <a href="#">floatingratebond.hpp</a> (Floating-rate bond ) . . . . .	1357
ql/Instruments/ <a href="#">forward.hpp</a> (Base forward class ) . . . . .	1358
ql/Instruments/ <a href="#">forwardrateagreement.hpp</a> (Forward rate agreement ) . . . . .	1359
ql/Instruments/ <a href="#">forwardvanillaoption.hpp</a> (Forward version of a vanilla option ) . . . . .	1360
ql/Instruments/ <a href="#">lookbackoption.hpp</a> (Lookback option on a single asset ) . . . . .	1361
ql/Instruments/ <a href="#">multiassetoption.hpp</a> (Option on multiple assets ) . . . . .	1362
ql/Instruments/ <a href="#">oneassetoption.hpp</a> (Option on a single asset ) . . . . .	1363
ql/Instruments/ <a href="#">oneassetstrikedoption.hpp</a> (Option on a single asset with striked payoff ) . . . . .	1364
ql/Instruments/ <a href="#">payoffs.hpp</a> (Payoffs for various options ) . . . . .	1365
ql/Instruments/ <a href="#">quantoforwardvanillaoption.hpp</a> (Quanto version of a forward vanilla option ) . . . . .	1366
ql/Instruments/ <a href="#">quantovanillaoption.hpp</a> (Quanto version of a vanilla option ) . . . . .	1367
ql/Instruments/ <a href="#">stock.hpp</a> (Concrete stock class ) . . . . .	1368
ql/Instruments/ <a href="#">swap.hpp</a> (Interest rate swap ) . . . . .	1369
ql/Instruments/ <a href="#">swaption.hpp</a> (Swaption class ) . . . . .	1370

ql/Instruments/ <a href="#">vanillaoption.hpp</a> (Vanilla option on a single asset ) . . . . .	1371
ql/Instruments/ <a href="#">vanillaswap.hpp</a> (Simple fixed-rate vs Libor swap ) . . . . .	1372
ql/Instruments/ <a href="#">varianceswap.hpp</a> (Variance Swap ) . . . . .	1373
ql/Instruments/ <a href="#">zerocouponbond.hpp</a> (Zero-coupon bond ) . . . . .	1374
ql/Lattices/ <a href="#">binomialtree.hpp</a> (Binomial tree class ) . . . . .	1376
ql/Lattices/ <a href="#">bsmllattice.hpp</a> (Binomial trees under the BSM model ) . . . . .	1378
ql/Lattices/ <a href="#">lattice.hpp</a> (Lattice method class ) . . . . .	1379
ql/Lattices/ <a href="#">lattice1d.hpp</a> (One-dimensional lattice class ) . . . . .	1380
ql/Lattices/ <a href="#">lattice2d.hpp</a> (Two-dimensional lattice class ) . . . . .	1381
ql/Lattices/ <a href="#">tflattice.hpp</a> (Binomial Tsiveriotis-Fernandes tree model ) . . . . .	1382
ql/Lattices/ <a href="#">tree.hpp</a> (Tree class ) . . . . .	1383
ql/Lattices/ <a href="#">trinomialtree.hpp</a> (Trinomial tree class ) . . . . .	1384
ql/MarketModels/ <a href="#">driftcalculator.hpp</a> (Drift computation for Market Model ) . . . . .	1385
ql/Math/ <a href="#">array.hpp</a> (1-D array used in linear algebra ) . . . . .	1386
ql/Math/ <a href="#">backwardflatinterpolation.hpp</a> (Backward-flat interpolation between discrete points ) . . . . .	1387
ql/Math/ <a href="#">beta.hpp</a> (Beta and beta incomplete functions ) . . . . .	1388
ql/Math/ <a href="#">bicubicsplineinterpolation.hpp</a> (Bicubic spline interpolation between discrete points ) . . . . .	1389
ql/Math/ <a href="#">bilinearinterpolation.hpp</a> (Bilinear interpolation between discrete points ) . . . . .	1390
ql/Math/ <a href="#">binomialdistribution.hpp</a> (Binomial distribution ) . . . . .	1391
ql/Math/ <a href="#">bivariatenormaldistribution.hpp</a> (Bivariate cumulative normal distribution ) . . . . .	1392
ql/Math/ <a href="#">chisquaredistribution.hpp</a> (Chi-square (central and non-central) distributions ) . . . . .	1393
ql/Math/ <a href="#">choleskydecomposition.hpp</a> (Cholesky decomposition ) . . . . .	1394
ql/Math/ <a href="#">comparison.hpp</a> (Floating-point comparisons ) . . . . .	1395
ql/Math/ <a href="#">convergencestatistics.hpp</a> (Statistics tool with risk measures ) . . . . .	1396
ql/Math/ <a href="#">cubicspline.hpp</a> (Cubic spline interpolation between discrete points ) . . . . .	1397
ql/Math/ <a href="#">discrepancystatistics.hpp</a> (Statistic tool for sequences with discrepancy calculation ) . . . . .	1398
ql/Math/ <a href="#">errorfunction.hpp</a> (Error function ) . . . . .	1399
ql/Math/ <a href="#">extrapolation.hpp</a> (Class-wide extrapolation settings ) . . . . .	1400
ql/Math/ <a href="#">factorial.hpp</a> (Factorial numbers calculator ) . . . . .	1401
ql/Math/ <a href="#">forwardflatinterpolation.hpp</a> (Forward-flat interpolation between discrete points ) . . . . .	1402
ql/Math/ <a href="#">functional.hpp</a> (Functionals and combinators not included in the STL ) . . . . .	1403
ql/Math/ <a href="#">gammadistribution.hpp</a> (Gamma distribution ) . . . . .	1404
ql/Math/ <a href="#">gaussianorthogonalpolynomial.hpp</a> (Orthogonal polynomials for gaussian quadratures ) . . . . .	1405
ql/Math/ <a href="#">gaussianquadratures.hpp</a> (Integral of a 1-dimensional function using the Gauss quadratures ) . . . . .	1406
ql/Math/ <a href="#">gaussianstatistics.hpp</a> (Statistics tool for gaussian-assumption risk measures ) . . . . .	1408
ql/Math/ <a href="#">generalstatistics.hpp</a> (Statistics tool ) . . . . .	1409
ql/Math/ <a href="#">incompletegamma.hpp</a> (Incomplete Gamma function ) . . . . .	1410
ql/Math/ <a href="#">incrementalstatistics.hpp</a> (Statistics tool based on incremental accumulation ) . . . . .	1411
ql/Math/ <a href="#">interpolation.hpp</a> (Base class for 1-D interpolations ) . . . . .	1412
ql/Math/ <a href="#">interpolation2D.hpp</a> (Abstract base classes for 2-D interpolations ) . . . . .	1413
ql/Math/ <a href="#">kronrodintegral.hpp</a> (Integral of a 1-dimensional function using the Gauss-Kronrod method ) . . . . .	1414
ql/Math/ <a href="#">lexicographicalview.hpp</a> (Lexicographical 2-D view of a contiguous set of data ) . . . . .	1415
ql/Math/ <a href="#">linearinterpolation.hpp</a> (Linear interpolation between discrete points ) . . . . .	1416
ql/Math/ <a href="#">linearleastquaresregression.hpp</a> (General linear least square regression ) . . . . .	1417
ql/Math/ <a href="#">loglinearinterpolation.hpp</a> (Log-linear interpolation between discrete points ) . . . . .	1418
ql/Math/ <a href="#">matrix.hpp</a> (Matrix used in linear algebra ) . . . . .	1419

ql/Math/ <a href="#">multicubicspline.hpp</a> (N-dimensional cubic spline interpolation between discrete points ) . . . . .	1420
ql/Math/ <a href="#">normaldistribution.hpp</a> (Normal, cumulative and inverse cumulative distributions ) . . . . .	1422
ql/Math/ <a href="#">poissondistribution.hpp</a> (Poisson distribution ) . . . . .	1423
ql/Math/ <a href="#">primenumbers.hpp</a> (Prime numbers calculator ) . . . . .	1424
ql/Math/ <a href="#">pseudosqrt.hpp</a> (Pseudo square root of a real symmetric matrix ) . . . . .	1425
ql/Math/ <a href="#">riskstatistics.hpp</a> (Empirical-distribution risk measures ) . . . . .	1426
ql/Math/ <a href="#">rounding.hpp</a> (Rounding implementation ) . . . . .	1427
ql/Math/ <a href="#">sabrinterpolation.hpp</a> (SABR interpolation interpolation between discrete points ) . . . . .	1428
ql/Math/ <a href="#">sampledcurve.hpp</a> (Class that contains a sampled curve ) . . . . .	1429
ql/Math/ <a href="#">segmentintegral.hpp</a> (Integral of a one-dimensional function ) . . . . .	1430
ql/Math/ <a href="#">sequencestatistics.hpp</a> (Statistics tools for sequence (vector, list, array) samples ) . . . . .	1431
ql/Math/ <a href="#">simpsonintegral.hpp</a> (Integral of a one-dimensional function ) . . . . .	1433
ql/Math/ <a href="#">statistics.hpp</a> (Statistics tool with risk measures ) . . . . .	1434
ql/Math/ <a href="#">svd.hpp</a> (Singular value decomposition ) . . . . .	1435
ql/Math/ <a href="#">symmetricschurdecomposition.hpp</a> (Eigenvalues / eigenvectors of a real symmetric matrix ) . . . . .	1436
ql/Math/ <a href="#">tqreigendecomposition.hpp</a> (Tridiag. QR eigen decomposition with explicit shift aka Wilkinson ) . . . . .	1437
ql/Math/ <a href="#">transformedgrid.hpp</a> (Encapsulates a grid ) . . . . .	1438
ql/Math/ <a href="#">trapezoidintegral.hpp</a> (Integral of a one-dimensional function ) . . . . .	1439
ql/MonteCarlo/ <a href="#">brownianbridge.hpp</a> (Browian bridge ) . . . . .	1441
ql/MonteCarlo/ <a href="#">earlyexercisepathpricer.hpp</a> (Base class for early exercise single-path pricers ) . . . . .	1442
ql/MonteCarlo/ <a href="#">getcovariance.hpp</a> (Covariance matrix calculation ) . . . . .	1443
ql/MonteCarlo/ <a href="#">longstaffschwartzpathpricer.hpp</a> (Longstaff-Schwarz path pricer for early exercise options ) . . . . .	1444
ql/MonteCarlo/ <a href="#">lsmbasisystem.hpp</a> (Utility classes for longstaff schwartz early exercise Monte Carlo ) . . . . .	1445
ql/MonteCarlo/ <a href="#">mctraits.hpp</a> (Monte Carlo policies ) . . . . .	1446
ql/MonteCarlo/ <a href="#">mctypedefs.hpp</a> (Default choices for template instantiations ) . . . . .	1447
ql/MonteCarlo/ <a href="#">montecarlomodel.hpp</a> (General purpose Monte Carlo model ) . . . . .	1448
ql/MonteCarlo/ <a href="#">multipath.hpp</a> (Correlated multiple asset paths ) . . . . .	1449
ql/MonteCarlo/ <a href="#">multipathgenerator.hpp</a> (Generates a multi path from a random-array generator ) . . . . .	1450
ql/MonteCarlo/ <a href="#">path.hpp</a> (Single factor random walk ) . . . . .	1451
ql/MonteCarlo/ <a href="#">pathgenerator.hpp</a> (Generates random paths using a sequence generator ) . . . . .	1452
ql/MonteCarlo/ <a href="#">pathpricer.hpp</a> (Base class for single-path pricers ) . . . . .	1453
ql/MonteCarlo/ <a href="#">sample.hpp</a> (Weighted sample ) . . . . .	1454
ql/Optimization/ <a href="#">armijo.hpp</a> (Armijo line-search class ) . . . . .	1456
ql/Optimization/ <a href="#">conjugategradient.hpp</a> (Conjugate gradient optimization method ) . . . . .	1457
ql/Optimization/ <a href="#">constraint.hpp</a> (Abstract constraint class ) . . . . .	1458
ql/Optimization/ <a href="#">costfunction.hpp</a> (Optimization cost function class ) . . . . .	1459
ql/Optimization/ <a href="#">criteria.hpp</a> (Optimization criteria class ) . . . . .	1460
ql/Optimization/ <a href="#">leastsquare.hpp</a> (Least square cost function ) . . . . .	1461
ql/Optimization/ <a href="#">levenbergmarquardt.hpp</a> (Levenberg-Marquardt optimization method ) . . . . .	1462
ql/Optimization/ <a href="#">linesearch.hpp</a> (Line search abstract class ) . . . . .	1463
ql/Optimization/ <a href="#">lmdif.hpp</a> (Wrapper for MINPACK minimization routine ) . . . . .	1464
ql/Optimization/ <a href="#">method.hpp</a> (Abstract optimization method class ) . . . . .	1465
ql/Optimization/ <a href="#">problem.hpp</a> (Abstract optimization class ) . . . . .	1466
ql/Optimization/ <a href="#">simplex.hpp</a> (Simplex optimization method ) . . . . .	1467
ql/Optimization/ <a href="#">steepestdescent.hpp</a> (Steepest descent optimization method ) . . . . .	1468

ql/Patterns/ <a href="#">bridge.hpp</a> (Bridge pattern (a.k.a. handle-body idiom) ) . . . . .	1470
ql/Patterns/ <a href="#">composite.hpp</a> (Composite pattern) . . . . .	1471
ql/Patterns/ <a href="#">curiouslyrecurring.hpp</a> (Curiously recurring template pattern) . . . . .	1472
ql/Patterns/ <a href="#">lazyobject.hpp</a> (Framework for calculation on demand and result caching) .	1473
ql/Patterns/ <a href="#">observable.hpp</a> (Observer/observable pattern) . . . . .	1474
ql/Patterns/ <a href="#">singleton.hpp</a> (Basic support for the singleton pattern) . . . . .	1475
ql/Patterns/ <a href="#">visitor.hpp</a> (Degenerate base class for the Acyclic Visitor pattern) . . . . .	1476
ql/Pricers/ <a href="#">discretegeometriccaso.hpp</a> (Discrete Geometric Average Strike Option) . . . .	1481
ql/Pricers/ <a href="#">mccliquetoption.hpp</a> (Cliquet option priced with Monte Carlo simulation) . .	1482
ql/Pricers/ <a href="#">mcdiscretearithmeticcaso.hpp</a> (Discrete Arithmetic Average Strike Option) . .	1483
ql/Pricers/ <a href="#">mceverest.hpp</a> (Everest-type option pricer) . . . . .	1484
ql/Pricers/ <a href="#">mchimalaya.hpp</a> (Himalayan-type option pricer) . . . . .	1485
ql/Pricers/ <a href="#">mcmaxbasket.hpp</a> (Max Basket Monte Carlo pricer) . . . . .	1486
ql/Pricers/ <a href="#">mcpagoda.hpp</a> (Roofed multi asset Asian option) . . . . .	1487
ql/Pricers/ <a href="#">mcperformanceoption.hpp</a> (Performance option priced with Monte Carlo sim- ulation) . . . . .	1488
ql/Pricers/ <a href="#">mcpricer.hpp</a> (Base class for Monte Carlo pricers) . . . . .	1489
ql/Pricers/ <a href="#">singleassetoption.hpp</a> (Common code for option evaluation) . . . . .	1490
ql/PricingEngines/ <a href="#">americanpayoffatexpiry.hpp</a> (Analytical formulae for american exer- cise with payoff at expiry) . . . . .	1493
ql/PricingEngines/ <a href="#">americanpayoffathit.hpp</a> (Analytical formulae for american exercise with payoff at hit) . . . . .	1494
ql/PricingEngines/ <a href="#">blackformula.hpp</a> (Black formula) . . . . .	1505
ql/PricingEngines/ <a href="#">blackmodel.hpp</a> (Black formula and associated functions) . . . . .	1506
ql/PricingEngines/ <a href="#">genericmodelengine.hpp</a> (Generic option engine based on a model) .	1518
ql/PricingEngines/ <a href="#">greeks.hpp</a> (Default greek calculations) . . . . .	1519
ql/PricingEngines/ <a href="#">latticeshortratemodelengine.hpp</a> (Engine for a short-rate model spe- cialized on a lattice) . . . . .	1522
ql/PricingEngines/ <a href="#">mclongstaffschwartzengine.hpp</a> (Longstaff Schwartz Monte Carlo en- gine for early exercise options) . . . . .	1525
ql/PricingEngines/ <a href="#">mcsimulation.hpp</a> (Framework for Monte Carlo engines) . . . . .	1526
ql/PricingEngines/Asian/ <a href="#">analytic_cont_geom_av_price.hpp</a> (Analytic engine for contin- uous geometric average price Asian) . . . . .	1495
ql/PricingEngines/Asian/ <a href="#">analytic_discr_geom_av_price.hpp</a> (Analytic engine for dis- crete geometric average price Asian) . . . . .	1496
ql/PricingEngines/Asian/ <a href="#">mc_discr_arith_av_price.hpp</a> (Monte Carlo engine for discrete arithmetic average price Asian) . . . . .	1497
ql/PricingEngines/Asian/ <a href="#">mc_discr_geom_av_price.hpp</a> (Monte Carlo engine for discrete geometric average price Asian) . . . . .	1498
ql/PricingEngines/Asian/ <a href="#">mcdiscreteasianengine.hpp</a> (Monte Carlo pricing engine for discrete average Asians) . . . . .	1499
ql/PricingEngines/Barrier/ <a href="#">analyticbarrierengine.hpp</a> (Analytic barrier option engines) .	1500
ql/PricingEngines/Barrier/ <a href="#">mcbarrierengine.hpp</a> (Monte Carlo barrier option engines) .	1501
ql/PricingEngines/Basket/ <a href="#">mcamericanbasketengine.hpp</a> (Least-square Monte Carlo en- gines) . . . . .	1502
ql/PricingEngines/Basket/ <a href="#">mcbasketengine.hpp</a> (European basket MC Engine) . . . . .	1503
ql/PricingEngines/Basket/ <a href="#">stulzengine.hpp</a> (2D European Basket formulae, due to Stulz (1982)) . . . . .	1504
ql/PricingEngines/CapFloor/ <a href="#">analyticcapfloorengine.hpp</a> (Analytic engine for caps/floors) .	1507
ql/PricingEngines/CapFloor/ <a href="#">blackcapfloorengine.hpp</a> (Black-formula cap/floor engine) .	1508
ql/PricingEngines/CapFloor/ <a href="#">discretizedcapfloor.hpp</a> (Discretized cap/floor) . . . . .	1509
ql/PricingEngines/CapFloor/ <a href="#">mchullwhiteengine.hpp</a> (Monte Carlo Hull-White engine for cap/floors) . . . . .	1510



ql/PricingEngines/CapFloor/ <a href="#">treecapfloorengine.hpp</a> (Numerical lattice engine for cap/floors ) . . . . .	1511
ql/PricingEngines/Cliquet/ <a href="#">analyticcliquetengine.hpp</a> (Analytic Cliquet engine ) . . . . .	1512
ql/PricingEngines/Cliquet/ <a href="#">analyticperformanceengine.hpp</a> (Analytic performance engine ) . . . . .	1513
ql/PricingEngines/Forward/ <a href="#">forwardengine.hpp</a> (Forward (strike-resetting) option engine ) . . . . .	1514
ql/PricingEngines/Forward/ <a href="#">forwardperformanceengine.hpp</a> (Forward (strike-resetting) performance option engines ) . . . . .	1515
ql/PricingEngines/Forward/ <a href="#">mcvarianceswapengine.hpp</a> (Monte Carlo variance-swap engine ) . . . . .	1516
ql/PricingEngines/Forward/ <a href="#">replicatingvarianceswapengine.hpp</a> (Replicating engine for variance swaps ) . . . . .	1517
ql/PricingEngines/Hybrid/ <a href="#">binomialconvertibleengine.hpp</a> (Binomial engine for convertible bonds ) . . . . .	1520
ql/PricingEngines/Hybrid/ <a href="#">discretizedconvertible.hpp</a> (Discretized convertible ) . . . . .	1521
ql/PricingEngines/Lookback/ <a href="#">analyticcontinuousfixedlookback.hpp</a> (Analytic engine for continuous fixed-strike lookback ) . . . . .	1523
ql/PricingEngines/Lookback/ <a href="#">analyticcontinuousfloatinglookback.hpp</a> (Analytic engine for continuous floating-strike lookback ) . . . . .	1524
ql/PricingEngines/Quanto/ <a href="#">quantoengine.hpp</a> (Quanto option engine ) . . . . .	1527
ql/PricingEngines/Swaption/ <a href="#">blackswaptionengine.hpp</a> (Black-formula swaption engine ) . . . . .	1528
ql/PricingEngines/Swaption/ <a href="#">discretizedswaption.hpp</a> (Discretized swaption class ) . . . . .	1529
ql/PricingEngines/Swaption/ <a href="#">g2swaptionengine.hpp</a> (Swaption pricing engine for two-factor additive Gaussian Model G2++ ) . . . . .	1530
ql/PricingEngines/Swaption/ <a href="#">jamshidianswaptionengine.hpp</a> (Swaption engine using Jamshidian's decomposition ) . . . . .	1531
ql/PricingEngines/Swaption/ <a href="#">lfmswaptionengine.hpp</a> (Libor forward model swaption engine based on black formula ) . . . . .	1532
ql/PricingEngines/Swaption/ <a href="#">treeswaptionengine.hpp</a> (Numerical lattice engines for swaps and swaptions ) . . . . .	1533
ql/PricingEngines/Vanilla/ <a href="#">analyticdigitalamericanengine.hpp</a> (Analytic digital American option engine ) . . . . .	1534
ql/PricingEngines/Vanilla/ <a href="#">analyticdividendeuropeanengine.hpp</a> (Analytic discrete-dividend European engine ) . . . . .	1535
ql/PricingEngines/Vanilla/ <a href="#">analyticeuropeanengine.hpp</a> (Analytic European engine ) . . . . .	1536
ql/PricingEngines/Vanilla/ <a href="#">analytichestonengine.hpp</a> (Analytic Heston-model engine ) . . . . .	1537
ql/PricingEngines/Vanilla/ <a href="#">baroneadesiwhaleyengine.hpp</a> (Barone-Adesi and Whaley approximation engine ) . . . . .	1538
ql/PricingEngines/Vanilla/ <a href="#">batesengine.hpp</a> (Analytic Bates model engine ) . . . . .	1539
ql/PricingEngines/Vanilla/ <a href="#">binomialengine.hpp</a> (Binomial option engine ) . . . . .	1540
ql/PricingEngines/Vanilla/ <a href="#">bjerkssundstenslandengine.hpp</a> (Bjerkssund and Stensland approximation engine ) . . . . .	1541
ql/PricingEngines/Vanilla/ <a href="#">discretizedvanillaoption.hpp</a> (Discretized vanilla option ) . . . . .	1542
ql/PricingEngines/Vanilla/ <a href="#">fdamericanengine.hpp</a> (Finite-differences American option engine ) . . . . .	1543
ql/PricingEngines/Vanilla/ <a href="#">fdbermudanengine.hpp</a> (Finite-difference Bermudan engine ) . . . . .	1544
ql/PricingEngines/Vanilla/ <a href="#">fdconditions.hpp</a> (Finite-difference templates to generate engines ) . . . . .	1545
ql/PricingEngines/Vanilla/ <a href="#">fddividendamericanengine.hpp</a> (American engine with discrete deterministic dividends ) . . . . .	1546
ql/PricingEngines/Vanilla/ <a href="#">fddividendengine.hpp</a> (Base engine for option with dividends ) . . . . .	1547
ql/PricingEngines/Vanilla/ <a href="#">fddividendeuropeanengine.hpp</a> (Finite-differences engine for European option with dividends ) . . . . .	1548

ql/PricingEngines/Vanilla/fddividendshoutengine.hpp (Base class for shout engine with dividends ) . . . . .	1549
ql/PricingEngines/Vanilla/fdeuropeanengine.hpp (Finite-difference European engine ) . . . . .	1550
ql/PricingEngines/Vanilla/fdmultiengine.hpp (Base engine for options with events happening at specific times ) . . . . .	1551
ql/PricingEngines/Vanilla/fdshoutengine.hpp (Finite-differences shout engine ) . . . . .	1552
ql/PricingEngines/Vanilla/fdstepconditionengine.hpp (Finite-differences step-condition engine ) . . . . .	1553
ql/PricingEngines/Vanilla/fdvanillaengine.hpp (Finite-differences vanilla-option engine ) . . . . .	1554
ql/PricingEngines/Vanilla/integralengine.hpp (Integral option engine ) . . . . .	1555
ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp (Jump diffusion (Merton 1976) engine ) . . . . .	1556
ql/PricingEngines/Vanilla/juquadraticengine.hpp (Ju quadratic (1999) approximation engine ) . . . . .	1557
ql/PricingEngines/Vanilla/mcamericanengine.hpp (American Monte Carlo engine ) . . . . .	1558
ql/PricingEngines/Vanilla/mcdigitalengine.hpp (Digital option Monte Carlo engine ) . . . . .	1559
ql/PricingEngines/Vanilla/mceuropeanengine.hpp (Monte Carlo European option engine ) . . . . .	1560
ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp (Monte Carlo Heston-model engine for European options ) . . . . .	1561
ql/PricingEngines/Vanilla/mcvanillaengine.hpp (Monte Carlo vanilla option engine ) . . . . .	1562
ql/Processes/blackscholesprocess.hpp (Black-Scholes processes ) . . . . .	1563
ql/Processes/eulerdiscretization.hpp (Euler discretization for stochastic processes ) . . . . .	1564
ql/Processes/forwardmeasureprocess.hpp (Forward-measure stochastic processes ) . . . . .	1565
ql/Processes/g2process.hpp (G2 stochastic processes ) . . . . .	1566
ql/Processes/geometricbrownianprocess.hpp (Geometric Brownian-motion process ) . . . . .	1567
ql/Processes/hestonprocess.hpp (Heston stochastic process ) . . . . .	1568
ql/Processes/hullwhiteprocess.hpp (Hull-White stochastic processes ) . . . . .	1569
ql/Processes/lfmcovarParams.hpp (Volatility & correlation function for libor forward model process ) . . . . .	1570
ql/Processes/lfmhullwhiteparams.hpp (Libor market model parameterization based on Hull White ) . . . . .	1571
ql/Processes/lfmprocess.hpp (Stochastic process of a libor forward model ) . . . . .	1572
ql/Processes/merton76process.hpp (Merton-76 process ) . . . . .	1573
ql/Processes/ornsteinuhlenbeckprocess.hpp (Ornstein-Uhlenbeck process ) . . . . .	1574
ql/Processes/squarerootprocess.hpp (Square-root process ) . . . . .	1575
ql/Processes/stochasticprocessarray.hpp (Array of correlated 1-D stochastic processes ) . . . . .	1576
ql/RandomNumbers/boxmullergaussianrng.hpp (Box-Muller Gaussian random-number generator ) . . . . .	1580
ql/RandomNumbers/centrallimitgaussianrng.hpp (Central limit Gaussian random-number generator ) . . . . .	1581
ql/RandomNumbers/faurersg.hpp (Faure low-discrepancy sequence generator ) . . . . .	1582
ql/RandomNumbers/haltonrng.hpp (Halton low-discrepancy sequence generator ) . . . . .	1583
ql/RandomNumbers/inversecumulativerng.hpp (Inverse cumulative Gaussian random-number generator ) . . . . .	1584
ql/RandomNumbers/inversecumulativesrg.hpp (Inverse cumulative random sequence generator ) . . . . .	1585
ql/RandomNumbers/knuthuniformrng.hpp (Knuth uniform random number generator ) . . . . .	1586
ql/RandomNumbers/lecuyeruniformrng.hpp (L'Ecuyer uniform random number generator ) . . . . .	1587
ql/RandomNumbers/mt19937uniformrng.hpp (Mersenne Twister uniform random number generator ) . . . . .	1588
ql/RandomNumbers/randomizedlds.hpp (Randomized low-discrepancy sequence ) . . . . .	1589

ql/RandomNumbers/ <a href="#">randomsequencegenerator.hpp</a> (Random sequence generator based on a pseudo-random number generator) . . . . .	1590
ql/RandomNumbers/ <a href="#">rngtraits.hpp</a> (Random-number generation policies) . . . . .	1591
ql/RandomNumbers/ <a href="#">seedgenerator.hpp</a> (Random seed generator) . . . . .	1593
ql/RandomNumbers/ <a href="#">sobolrsg.hpp</a> (Sobol low-discrepancy sequence generator) . . . . .	1594
ql/ShortRateModels/ <a href="#">calibrationhelper.hpp</a> (Calibration helper class) . . . . .	1597
ql/ShortRateModels/ <a href="#">model.hpp</a> (Abstract interest rate model class) . . . . .	1612
ql/ShortRateModels/ <a href="#">onefactormodel.hpp</a> (Abstract one-factor interest rate model class) . . . . .	1613
ql/ShortRateModels/ <a href="#">parameter.hpp</a> (Model parameter classes) . . . . .	1619
ql/ShortRateModels/ <a href="#">twofactormodel.hpp</a> (Abstract two-factor interest rate model class) . . . . .	1620
ql/ShortRateModels/CalibrationHelpers/ <a href="#">caphelper.hpp</a> (CapHelper calibration helper) . . . . .	1598
ql/ShortRateModels/CalibrationHelpers/ <a href="#">hestonmodelhelper.hpp</a> (Heston-model calibration helper) . . . . .	1599
ql/ShortRateModels/CalibrationHelpers/ <a href="#">swaptionhelper.hpp</a> (Swaption calibration helper) . . . . .	1600
ql/ShortRateModels/LiborMarketModels/ <a href="#">lfmcovarproxy.hpp</a> (Proxy for libor forward covariance parameterization) . . . . .	1601
ql/ShortRateModels/LiborMarketModels/ <a href="#">liborforwardmodel.hpp</a> (Libor forward model incl. exact cap pricing Rebonato formula to approximate swaption prices) . . . . .	1602
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmconstwrappercorrmodel.hpp</a> (Const wrapper for correlation model for libor market models) . . . . .	1603
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmconstwrappervolmodel.hpp</a> (Const wrapper for a volatility model for libor market models) . . . . .	1604
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmcrrmodel.hpp</a> (Correlation model for libor market models) . . . . .	1605
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmexpcorrmodel.hpp</a> (Exponential correlation model for libor market models) . . . . .	1606
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmextlinexpvolmodel.hpp</a> (Volatility model for libor market models) . . . . .	1607
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmfixedvolmodel.hpp</a> (Model of constant volatilities for libor market models) . . . . .	1608
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmlinexpcorrmodel.hpp</a> (Exponential correlation model for libor market models) . . . . .	1609
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmlinexpvolmodel.hpp</a> (Volatility model for libor market models) . . . . .	1610
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmvolmodel.hpp</a> (Volatility model for libor market models) . . . . .	1611
ql/ShortRateModels/OneFactorModels/ <a href="#">blackkarasinski.hpp</a> (Black-Karasinski model) . . . . .	1614
ql/ShortRateModels/OneFactorModels/ <a href="#">coxingersollross.hpp</a> (Cox-Ingersoll-Ross model) . . . . .	1615
ql/ShortRateModels/OneFactorModels/ <a href="#">extendedcoxingersollross.hpp</a> (Extended Cox-Ingersoll-Ross model) . . . . .	1616
ql/ShortRateModels/OneFactorModels/ <a href="#">hullwhite.hpp</a> (Hull & White (HW) model) . . . . .	1617
ql/ShortRateModels/OneFactorModels/ <a href="#">vasicek.hpp</a> (Vasicek model class) . . . . .	1618
ql/ShortRateModels/TwoFactorModels/ <a href="#">batesmodel.hpp</a> (Extended versions of the Heston model) . . . . .	1621
ql/ShortRateModels/TwoFactorModels/ <a href="#">g2.hpp</a> (Two-factor additive Gaussian Model G2++) . . . . .	1622
ql/ShortRateModels/TwoFactorModels/ <a href="#">hestonmodel.hpp</a> (Heston model for the stochastic volatility of an asset) . . . . .	1623
ql/Solvers1D/ <a href="#">bisection.hpp</a> (Bisection 1-D solver) . . . . .	1625
ql/Solvers1D/ <a href="#">brent.hpp</a> (Brent 1-D solver) . . . . .	1626
ql/Solvers1D/ <a href="#">falseposition.hpp</a> (False-position 1-D solver) . . . . .	1627
ql/Solvers1D/ <a href="#">newton.hpp</a> (Newton 1-D solver) . . . . .	1628
ql/Solvers1D/ <a href="#">newtonsafe.hpp</a> (Safe (bracketed) Newton 1-D solver) . . . . .	1629



ql/Solvers1D/ <a href="#">ridder.hpp</a> (Ridder 1-D solver) . . . . .	1630
ql/Solvers1D/ <a href="#">secant.hpp</a> (Secant 1-D solver) . . . . .	1631
ql/TermStructures/ <a href="#">bondhelpers.hpp</a> (Bond rate helpers) . . . . .	1635
ql/TermStructures/ <a href="#">bootstraptraits.hpp</a> (Bootstrap traits) . . . . .	1636
ql/TermStructures/ <a href="#">compoundforward.hpp</a> (Compounded forward term structure) . . . . .	1637
ql/TermStructures/ <a href="#">discountcurve.hpp</a> (Interpolated discount factor structure) . . . . .	1638
ql/TermStructures/ <a href="#">drifttermstructure.hpp</a> (Drift term structure) . . . . .	1639
ql/TermStructures/ <a href="#">extendeddiscountcurve.hpp</a> (Discount factor structure with detailed compound-forward calculation) . . . . .	1640
ql/TermStructures/ <a href="#">flatforward.hpp</a> (Flat forward rate term structure) . . . . .	1641
ql/TermStructures/ <a href="#">forwardcurve.hpp</a> (Interpolated forward-rate structure) . . . . .	1642
ql/TermStructures/ <a href="#">forwardspreadedtermstructure.hpp</a> (Forward-spreaded term struc- ture) . . . . .	1643
ql/TermStructures/ <a href="#">forwardstructure.hpp</a> (Forward-based yield term structure) . . . . .	1644
ql/TermStructures/ <a href="#">impliedtermstructure.hpp</a> (Implied term structure) . . . . .	1645
ql/TermStructures/ <a href="#">piecewiseflatforward.hpp</a> (Piecewise flat forward term structure) . . . . .	1646
ql/TermStructures/ <a href="#">piecewiseyieldcurve.hpp</a> (Piecewise-interpolated term structure) . . . . .	1647
ql/TermStructures/ <a href="#">piecewisezerospreadedtermstructure.hpp</a> (Piecewise-zero-spreaded term structure) . . . . .	1648
ql/TermStructures/ <a href="#">quantotermstructure.hpp</a> (Quanto term structure) . . . . .	1649
ql/TermStructures/ <a href="#">ratehelpers.hpp</a> (Deposit, FRA, futures, and swap rate helpers) . . . . .	1650
ql/TermStructures/ <a href="#">zerocurve.hpp</a> (Interpolated zero-rates structure) . . . . .	1651
ql/TermStructures/ <a href="#">zerospreadedtermstructure.hpp</a> (Zero spreaded term structure) . . . . .	1652
ql/TermStructures/ <a href="#">zeroyieldstructure.hpp</a> (Zero-yield based term structure) . . . . .	1653
ql/Utilities/ <a href="#">clone.hpp</a> (Cloning proxy to an underlying object) . . . . .	1658
ql/Utilities/ <a href="#">dataformatters.hpp</a> (Output manipulators) . . . . .	1659
ql/Utilities/ <a href="#">dataparsers.hpp</a> (Classes used to parse data for input) . . . . .	1660
ql/Utilities/ <a href="#">disposable.hpp</a> (Generic disposable object with move semantics) . . . . .	1661
ql/Utilities/ <a href="#">null.hpp</a> (Null values) . . . . .	1662
ql/Utilities/ <a href="#">observablevalue.hpp</a> (Observable and assignable proxy to concrete value) . . . . .	1663
ql/Utilities/ <a href="#">steppingiterator.hpp</a> (Iterator advancing in constant steps) . . . . .	1664
ql/Utilities/ <a href="#">strings.hpp</a> (String utilities) . . . . .	1665
ql/Utilities/ <a href="#">tracing.hpp</a> (Tracing facilities) . . . . .	1666
ql/Volatilities/ <a href="#">blackconstantvol.hpp</a> (Black constant volatility, no time dependence, no strike dependence) . . . . .	1668
ql/Volatilities/ <a href="#">blackvariancecurve.hpp</a> (Black volatility curve modelled as variance curve) . . . . .	1669
ql/Volatilities/ <a href="#">blackvariancesurface.hpp</a> (Black volatility surface modelled as variance surface) . . . . .	1670
ql/Volatilities/ <a href="#">capflatvolvector.hpp</a> (Cap/floor at-the-money flat volatility vector) . . . . .	1671
ql/Volatilities/ <a href="#">capletconstantvol.hpp</a> (Constant caplet volatility) . . . . .	1672
ql/Volatilities/ <a href="#">capletvariancecurve.hpp</a> (Caplet variance curve) . . . . .	1673
ql/Volatilities/ <a href="#">cmsmarket.hpp</a> . . . . .	1674
ql/Volatilities/ <a href="#">impliedvoltermstructure.hpp</a> (Implied Black Vol Term Structure) . . . . .	1675
ql/Volatilities/ <a href="#">localconstantvol.hpp</a> (Local constant volatility, no time dependence, no asset dependence) . . . . .	1676
ql/Volatilities/ <a href="#">localvolcurve.hpp</a> (Local volatility curve derived from a Black curve) . . . . .	1677
ql/Volatilities/ <a href="#">localvolsurface.hpp</a> (Local volatility surface derived from a Black vol sur- face) . . . . .	1678
ql/Volatilities/ <a href="#">smilesection.hpp</a> (Swaption volatility structure) . . . . .	1679
ql/Volatilities/ <a href="#">swaptionconstantvol.hpp</a> (Constant swaption volatility) . . . . .	1680
ql/Volatilities/ <a href="#">swaptionvolcube.hpp</a> (Swaption volatility cube) . . . . .	1681
ql/Volatilities/ <a href="#">swaptionvolcubebyabr.hpp</a> (Swaption volatility cube by Sabr) . . . . .	1682
ql/Volatilities/ <a href="#">swaptionvolmatrix.hpp</a> (Swaption at-the-money volatility matrix) . . . . .	1683
ql/VolatilityModels/ <a href="#">constantestimator.hpp</a> (Constant volatility estimator) . . . . .	1685

ql/VolatilityModels/ <a href="#">garch.hpp</a> (GARCH volatility model) . . . . .	1686
ql/VolatilityModels/ <a href="#">garmanklass.hpp</a> (Volatility estimators using high low data) . . . .	1687
ql/VolatilityModels/ <a href="#">simplelocalestimator.hpp</a> (Constant volatility estimator) . . . . .	1688

## Chapter 6

# QuantLib Module Documentation

### 6.1 Numeric types

#### 6.1.1 Detailed Description

A number of numeric types are defined in order to add clarity to function and method declarations.

#### Typedefs

- typedef QL\_INTEGER [Integer](#)  
*integer number*
- typedef QL\_BIG\_INTEGER [BigInteger](#)  
*large integer number*
- typedef unsigned QL\_INTEGER [Natural](#)  
*positive integer*
- typedef QL\_REAL [Real](#)  
*real number*
- typedef Real [Decimal](#)  
*decimal number*
- typedef std::size\_t [Size](#)  
*size of a container*
- typedef Real [Time](#)  
*continuous quantity with 1-year units*
- typedef Real [DiscountFactor](#)  
*discount factor between dates*
- typedef Real [Rate](#)  
*interest rates*

- typedef Real [Spread](#)  
*spreads on interest rates*
- typedef Real [Volatility](#)  
*volatility*

## 6.2 Currencies and FX rates

### Classes

- class [ZARCurrency](#)  
*South-African rand.*
- class [ARSCurrency](#)  
*Argentinian peso.*
- class [BRLCurrency](#)  
*Brazilian real.*
- class [CADCurrency](#)  
*Canadian dollar.*
- class [CLPCurrency](#)  
*Chilean peso.*
- class [COPCurrency](#)  
*Colombian peso.*
- class [MXNCurrency](#)  
*Mexican peso.*
- class [TTDCurrency](#)  
*Trinidad & Tobago dollar.*
- class [USDCurrency](#)  
*U.S. dollar.*
- class [VEBCurrency](#)  
*Venezuelan bolivar.*
- class [BDTCurrency](#)  
*Bangladesh taka.*
- class [CNYCurrency](#)  
*Chinese yuan.*
- class [HKDCurrency](#)  
*Honk Kong dollar.*
- class [ILSCurrency](#)  
*Israeli shekel.*
- class [INRCurrency](#)  
*Indian rupee.*
- class [IQDCurrency](#)

*Iraqi dinar.*

- class [IRRCurrency](#)  
*Iranian rial.*
- class [JPYCurrency](#)  
*Japanese yen.*
- class [KRWCurrency](#)  
*South-Korean won.*
- class [KWDCurrency](#)  
*Kuwaiti dinar.*
- class [NPRCurrency](#)  
*Nepal rupee.*
- class [PKRCurrency](#)  
*Pakistani rupee.*
- class [SARCurrency](#)  
*Saudi riyal.*
- class [SGDCurrency](#)  
*Singapore dollar.*
- class [THBCurrency](#)  
*Thai baht.*
- class [TWDCurrency](#)  
*Taiwan dollar.*
- class [BGLCurrency](#)  
*Bulgarian lev.*
- class [BYRCurrency](#)  
*Belarussian ruble.*
- class [CHFCurrency](#)  
*Swiss franc.*
- class [CYPCurrency](#)  
*Cyprus pound.*
- class [CZKCurrency](#)  
*Czech koruna.*
- class [DKKCurrency](#)  
*Danish krone.*

- class [EEKCurrency](#)  
*Estonian kroon.*
- class [EURCurrency](#)  
*European Euro.*
- class [GBPCurrency](#)  
*British pound sterling.*
- class [HUFCurrency](#)  
*Hungarian forint.*
- class [ISKCurrency](#)  
*Iceland krona.*
- class [LTLCurrency](#)  
*Lithuanian litas.*
- class [LVLCurrency](#)  
*Latvian lat.*
- class [MTLCurrency](#)  
*Maltese lira.*
- class [NOKCurrency](#)  
*Norwegian krone.*
- class [PLNCurrency](#)  
*Polish zloty.*
- class [ROLCurrency](#)  
*Romanian leu.*
- class [RONCurrency](#)  
*Romanian new leu.*
- class [SEKCurrenecy](#)  
*Swedish krona.*
- class [SITCurrency](#)  
*Slovenian tolar.*
- class [SKKCurrenecy](#)  
*Slovak koruna.*
- class [TRLCurrency](#)  
*Turkish lira.*
- class [TRYCurrency](#)  
*New Turkish lira.*

- class [ATSCurrency](#)  
*Austrian shilling.*
- class [BEFCurrency](#)  
*Belgian franc.*
- class [DEMCurrency](#)  
*Deutsche mark.*
- class [ESPCurrency](#)  
*Spanish peseta.*
- class [FIMCurrency](#)  
*Finnish markka.*
- class [FRFCurrency](#)  
*French franc.*
- class [GRDCurrency](#)  
*Greek drachma.*
- class [IEPCurrency](#)  
*Irish punt.*
- class [ITLCurrency](#)  
*Italian lira.*
- class [LUFCurrency](#)  
*Luxembourg franc.*
- class [NLGCurrency](#)  
*Dutch guilder.*
- class [PTECurrency](#)  
*Portuguese escudo.*
- class [AUDCurrency](#)  
*Australian dollar.*
- class [NZDCurrency](#)  
*New Zealand dollar.*



## 6.3 Date and time calculations

### 6.3.1 Detailed Description

The concrete class `QuantLib::Date` 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.

### Modules

- `Calendars`
- `Day counters`

### Classes

- class `Calendar`  
*calendar class*
- class `Date`  
*Concrete date class.*
- class `DayCounter`  
*day counter class*
- class `Period`  
*Time period described by a number of a given time unit.*

### Typedefs

- typedef Integer `Day`  
*Day number.*
- typedef Integer `Year`  
*Year number.*

## Enumerations

- enum `BusinessDayConvention` {  
`Following`, `ModifiedFollowing`, `Preceding`, `ModifiedPreceding`,  
`Unadjusted`, `MonthEndReference`, `UnadjustedMonthEnd` }  
*Business Day conventions.*
- enum `Weekday` {  
`Sunday` = 1, `Monday` = 2, `Tuesday` = 3, `Wednesday` = 4,  
`Thursday` = 5, `Friday` = 6, `Saturday` = 7, `Sun` = 1,  
`Mon` = 2, `Tue` = 3, `Wed` = 4, `Thu` = 5,  
`Fri` = 6, `Sat` = 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,  
`Jan` = 1, `Feb` = 2, `Mar` = 3, `Apr` = 4,  
`Jun` = 6, `Jul` = 7, `Aug` = 8, `Sep` = 9,  
`Oct` = 10, `Nov` = 11, `Dec` = 12 }  
*Month names.*
- enum `Frequency` {  
`NoFrequency` = -1, `Once` = 0, `Annual` = 1, `Semiannual` = 2,  
`EveryFourthMonth` = 3, `Quarterly` = 4, `Bimonthly` = 6, `Monthly` = 12,  
`Biweekly` = 26, `Weekly` = 52, `Daily` = 365 }  
*Frequency of events.*
- enum `TimeUnit` { `Days`, `Weeks`, `Months`, `Years` }  
*Units used to describe time periods.*

## 6.3.2 Enumeration Type Documentation

### 6.3.2.1 enum BusinessDayConvention

Business Day conventions.

These conventions specify the algorithm used to adjust a date in case it is not a valid business day.

#### Enumerator:

*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.

*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.

*Unadjusted* Do not adjust.

*MonthEndReference* Choose the first business day after the given holiday. If the original date falls on last business day of the month, choose the last business day of the month

*UnadjustedMonthEnd* Do not adjust, unless the original date falls on the last business day of month. In this case, choose the last day of the month regardless of whether it is a business day.

#### 6.3.2.2 enum Weekday

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

#### 6.3.2.3 enum Frequency

Frequency of events.

Enumerator:

*NoFrequency* null frequency

*Once* only once, e.g., a zero-coupon

*Annual* once a year

*Semiannual* twice a year

*EveryFourthMonth* every fourth month

*Quarterly* every third month

*Bimonthly* every second month

*Monthly* once a month

*Biweekly* every second week

*Weekly* once a week

*Daily* once a day

## 6.4 Calendars

### 6.4.1 Detailed Description

The class `QuantLib::Calendar` provides the interface for determining whether a date is a business day or a holiday for a given exchange or a given country, and for incrementing/decrementing a date of a given number of business days. A number of calendars is contained in the `ql/Calendars` directory.

#### Classes

- class `Argentina`  
*Argentinian calendars.*
- class `Australia`  
*Australian calendar.*
- class `Brazil`  
*Brazilian calendar.*
- class `Canada`  
*Canadian calendar.*
- class `China`  
*Chinese calendar.*
- class `CzechRepublic`  
*Czech calendars.*
- class `Denmark`  
*Danish calendar.*
- class `Finland`  
*Finnish calendar.*
- class `Germany`  
*German calendars.*
- class `HongKong`  
*Hong Kong calendars.*
- class `Hungary`  
*Hungarian calendar.*
- class `Iceland`  
*Icelandic calendars.*
- class `India`  
*Indian calendars.*

- class [Indonesia](#)  
*Indonesian calendars*
- class [Italy](#)  
*Italian calendars.*
- class [Japan](#)  
*Japanese calendar.*
- class [JointCalendar](#)  
*Joint calendar.*
- class [Mexico](#)  
*Mexican calendars*
- class [NewZealand](#)  
*New Zealand calendar.*
- class [Norway](#)  
*Norwegian calendar.*
- class [NullCalendar](#)  
*Calendar for reproducing theoretical calculations.*
- class [Poland](#)  
*Polish calendar.*
- class [SaudiArabia](#)  
*Saudi Arabian calendar.*
- class [Singapore](#)  
*Singapore calendars*
- class [Slovakia](#)  
*Slovak calendars.*
- class [SouthAfrica](#)  
*South-African calendar.*
- class [SouthKorea](#)  
*South Korean calendars.*
- class [Sweden](#)  
*Swedish calendar.*
- class [Switzerland](#)  
*Swiss calendar.*
- class [Taiwan](#)

*Taiwanese calendars.*

- class [TARGET](#)  
*TARGET calendar*
- class [Turkey](#)  
*Turkish calendar.*
- class [Ukraine](#)  
*Ukrainian calendars.*
- class [UnitedKingdom](#)  
*United Kingdom calendars.*
- class [UnitedStates](#)  
*United States calendars.*

## 6.5 Day counters

### 6.5.1 Detailed Description

The class `QuantLib::DayCounter` provides more advanced means of measuring the distance between two dates according to a given market convention, both as number of days or fraction of year. A number of such conventions is contained in the `ql/DayCounters` directory.

#### Classes

- class `Actual360`  
*Actual/360 day count convention.*
- class `Actual365Fixed`  
*Actual/365 (Fixed) day count convention.*
- class `ActualActual`  
*Actual/Actual day count.*
- class `Business252`  
*Business/252 day count convention.*
- class `OneDayCounter`  
*1/1 day count convention*
- class `SimpleDayCounter`  
*Simple day counter for reproducing theoretical calculations.*
- class `Thirty360`  
*30/360 day count convention*

## 6.6 Pricing engines

### Modules

- [Asian option engines](#)
- [Barrier option engines](#)
- [Basket option engines](#)
- [Cap/floor engines](#)
- [Cliquet option engines](#)
- [Forward option engines](#)
- [Quanto option engines](#)
- [Swaption engines](#)
- [Vanilla option engines](#)



## 6.7 Asian option engines

### Classes

- class [AnalyticContinuousGeometricAveragePriceAsianEngine](#)  
*Pricing engine for European continuous geometric average price Asian.*
- class [AnalyticDiscreteGeometricAveragePriceAsianEngine](#)  
*Pricing engine for European discrete geometric average price Asian.*
- class [MCDiscreteArithmeticAPEngine](#)  
*Monte Carlo pricing engine for discrete arithmetic average price Asian.*
- class [MCDiscreteGeometricAPEngine](#)  
*Monte Carlo pricing engine for discrete geometric average price Asian.*
- class [MCDiscreteAveragingAsianEngine](#)  
*Pricing engine for discrete average Asians using Monte Carlo simulation.*

## 6.8 Barrier option engines

### Classes

- class [AnalyticBarrierEngine](#)  
*Pricing engine for barrier options using analytical formulae.*
- class [MCBarrierEngine](#)  
*Pricing engine for barrier options using Monte Carlo simulation.*

## 6.9 Basket option engines

### Classes

- class [MCAmericanBasketEngine](#)  
*least-square Monte Carlo engine*
- class [MCBasketEngine](#)  
*Pricing engine for basket options using Monte Carlo simulation.*
- class [StulzEngine](#)  
*Pricing engine for 2D European Baskets.*

## 6.10 Cap/floor engines

### Classes

- class [AnalyticCapFloorEngine](#)  
*Analytic engine for cap/floor.*
- class [BlackCapFloorEngine](#)  
*Black-formula cap/floor engine.*
- class [MCHullWhiteCapFloorEngine](#)  
*Monte Carlo Hull-White engine for cap/floors.*
- class [TreeCapFloorEngine](#)  
*Numerical lattice engine for cap/floors.*

## 6.11 Cliquet option engines

### Classes

- class [AnalyticCliquetEngine](#)  
*Pricing engine for Cliquet options using analytical formulae.*
- class [AnalyticPerformanceEngine](#)  
*Pricing engine for performance options using analytical formulae.*

## 6.12 Forward option engines

### Classes

- class [ForwardEngine](#)  
*Forward engine base class.*
- class [ForwardPerformanceEngine](#)  
*Forward performance engine.*
- class [MCVarianceSwapEngine](#)  
*Variance-swap pricing engine using Monte Carlo simulation,.*
- class [ReplicatingVarianceSwapEngine](#)  
*Variance-swap pricing engine using replicating cost,.*

## 6.13 Quanto option engines

### Classes

- class [QuantoEngine](#)  
*Quanto engine base class.*

## 6.14 Swaption engines

### Classes

- class [BlackSwaptionEngine](#)  
*Black-formula swaption engine.*
- class [G2SwaptionEngine](#)  
*Swaption priced by means of the Black formula*
- class [JamshidianSwaptionEngine](#)  
*Jamshidian swaption engine.*
- class [LfmSwaptionEngine](#)  
*libor forward model swaption engine based on black formula*
- class [TreeSwaptionEngine](#)  
*Numerical lattice engine for swaptions.*



## 6.15 Vanilla option engines

### Classes

- class [AnalyticDigitalAmericanEngine](#)
- class [AnalyticDividendEuropeanEngine](#)  
*Analytic pricing engine for European options with discrete dividends.*
- class [AnalyticEuropeanEngine](#)  
*Pricing engine for European vanilla options using analytical formulae.*
- class [AnalyticHestonEngine](#)  
*analytic Heston-model engine based on Fourier transform*
- class [BaroneAdesiWhaleyApproximationEngine](#)
- class [BatesEngine](#)  
*Bates model engines based on Fourier transform.*
- class [BinomialVanillaEngine](#)  
*Pricing engine for vanilla options using binomial trees.*
- class [Bjerk SundStenslandApproximationEngine](#)
- class [FDBermudanEngine](#)  
*Finite-differences Bermudan engine.*
- class [FDDividendEngineMerton73](#)  
*Finite-differences pricing engine for dividend options using.*
- class [FDDividendEngineShiftScale](#)  
*Finite-differences pricing engine for dividend options using.*
- class [FDEuropeanEngine](#)  
*Pricing engine for European options using finite-differences.*
- class [FDStepConditionEngine](#)  
*Finite-differences pricing engine for American-style vanilla options.*
- class [IntegralEngine](#)
- class [JumpDiffusionEngine](#)  
*Jump-diffusion engine for vanilla options.*
- class [JuQuadraticApproximationEngine](#)
- class [MCAmericanEngine](#)  
*American Monte Carlo engine.*
- class [MCDigitalEngine](#)  
*Pricing engine for digital options using Monte Carlo simulation.*
- class [MCEuropeanEngine](#)  
*European option pricing engine using Monte Carlo simulation.*

- class [MCEuropeanHestonEngine](#)  
*Monte Carlo Heston-model engine for European options.*
- class [MCVanillaEngine](#)  
*Pricing engine for vanilla options using Monte Carlo simulation.*

## Typedefs

- typedef `FDEngineAdapter< FDAmericanCondition< FDStepConditionEngine >, OneAssetOption::engine >` [FDAmericanEngine](#)  
*Finite-differences pricing engine for American one asset options.*
- typedef `FDEngineAdapter< FDAmericanCondition< FDDividendEngine >, DividendVanillaOption::engine >` [FDDividendAmericanEngine](#)  
*Finite-differences pricing engine for dividend American options.*
- typedef `FDEngineAdapter< FDDividendEngine, DividendVanillaOption::engine >` [FDDividendEuropeanEngine](#)  
*Finite-differences pricing engine for dividend European options.*
- typedef `FDEngineAdapter< FDShoutCondition< FDDividendEngine >, DividendVanillaOption::engine >` [FDDividendShoutEngine](#)  
*Finite-differences shout engine with dividends.*
- typedef `FDEngineAdapter< FDShoutCondition< FDStepConditionEngine >, VanillaOption::engine >` [FDShoutEngine](#)  
*Finite-differences pricing engine for shout vanilla options.*

### 6.15.1 Typedef Documentation

#### 6.15.1.1 `typedef FDEngineAdapter<FDAmericanCondition<FDStepConditionEngine>, OneAssetOption::engine> FDAmericanEngine`

Finite-differences pricing engine for American one asset options.

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

#### Examples:

[EquityOption.cpp](#).

**6.15.1.2** `typedef FDEngineAdapter<FDAmericanCondition<FDDividendEngine>, DividendVanillaOption::engine> FDDividendAmericanEngine`

Finite-differences pricing engine for dividend American options.

#### Tests

- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the invariance of the results upon addition of null dividends is tested.

#### Bug

results are not overly reliable.

#### Bug

method `impliedVolatility()` utterly fails

**6.15.1.3** `typedef FDEngineAdapter<FDDividendEngine, DividendVanillaOption::engine> FDDividendEuropeanEngine`

Finite-differences pricing engine for dividend European options.

#### Tests

- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the invariance of the results upon addition of null dividends is tested.

**6.15.1.4** `typedef FDEngineAdapter<FDShoutCondition<FDDividendEngine>, DividendVanillaOption::engine> FDDividendShoutEngine`

Finite-differences shout engine with dividends.

#### Bug

results are not overly reliable.

**6.15.1.5** `typedef FDEngineAdapter<FDShoutCondition<FDStepConditionEngine>, VanillaOption::engine> FDShoutEngine`

Finite-differences pricing engine for shout vanilla options.

#### Tests

the correctness of the returned greeks is tested by reproducing numerical derivatives.

## 6.16 Finite-differences framework

### 6.16.1 Detailed Description

This framework (corresponding to the `ql/FiniteDifferences` directory) contains basic building blocks for the numerical solution of partial differential equations by means of finite-difference methods.

#### Classes

- 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 [OperatorFactory](#)  
*Black-Scholes-Merton differential operator.*
- class [StepConditionSet](#)  
*Parallel evolver for multiple arrays.*
- class [StepCondition](#)  
*condition to be applied at every time step*
- class [NullCondition](#)  
*null step condition*
- class [TridiagonalOperator](#)  
*Base implementation for tridiagonal operator.*

## Typedefs

- typedef `PdeOperator< PdeBSM >` [BSMTermOperator](#)  
*Black-Scholes-Merton differential operator.*
- typedef `PdeOperator< PdeShortRate >` [OneFactorOperator](#)  
*Interest-rate single factor model differential operator.*

### 6.16.2 Typedef Documentation

#### 6.16.2.1 typedef `PdeOperator<PdeBSM>` [BSMTermOperator](#)

Black-Scholes-Merton differential operator.

#### Tests

coefficients are tested against constant BSM operator

## 6.17 Short-rate modelling framework

### 6.17.1 Detailed Description

This framework (corresponding to the `ql/ShortRateModels` directory) implements some single-factor and two-factor short rate models. The models implemented in this library are widely used by practitioners. 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::AffineModel](#) class), analytical formulas for discount bonds and discount bond options are given (useful for calibration).

### 6.17.2 Single-factor models

#### The Hull & White model

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

When  $\alpha$  and  $\sigma$  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).

### 6.17.3 Calibration

The class `CalibrationHelper` is a base class that facilitates the instantiation 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 classes are [QuantLib::CapHelper](#) and [QuantLib::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 `ql/Optimization` directory.

### 6.17.4 Two-factor models

### 6.17.5 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::AnalyticalCapFloor`. In the case of single-factor affine models, swaptions can be priced using the Jamshidian decomposition, implemented in `QuantLib::JamshidianSwaption`.

#### 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::OneFactorOperator` 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-independent derivative. Just implement a class derived from `NumericalDerivative` (see `QuantLib::NumericalSwaption` for example) and roll it back until the present time... Just look at `QuantLib::TreeCapFloor` and `QuantLib::TreeSwaption` for working pricers.

### Classes

- class [AffineModel](#)  
*Affine model class.*
- class [TermStructureConsistentModel](#)  
*Term-structure consistent model class.*
- class [ShortRateModel](#)  
*Abstract short-rate model class.*

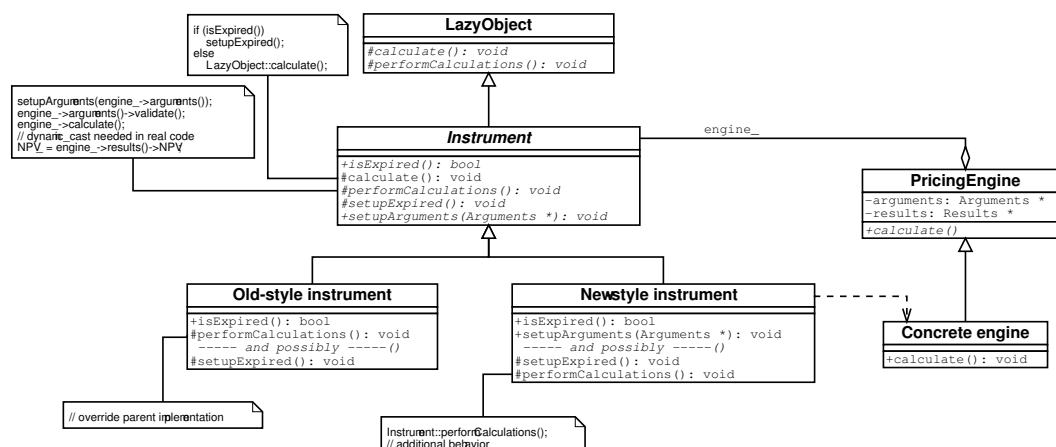
- class [OneFactorModel](#)  
*Single-factor short-rate model abstract class.*
- class [OneFactorAffineModel](#)  
*Single-factor affine base class.*
- class [BlackKarasinski](#)  
*Standard Black-Karasinski model class.*
- class [CoxIngersollRoss](#)  
*Cox-Ingersoll-Ross model class.*
- class [ExtendedCoxIngersollRoss](#)  
*Extended Cox-Ingersoll-Ross model class.*
- class [HullWhite](#)  
*Single-factor Hull-White (extended Vasicek) model class.*
- class [Vasicek](#)  
*Vasicek model class*
- class [TwoFactorModel](#)  
*Abstract base-class for two-factor models.*
- class [G2](#)  
*Two-additive-factor gaussian model class.*



## 6.18 Financial instruments

### 6.18.1 Detailed Description

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.

## Classes

- class [ContinuousAveragingAsianOption](#)

*Continuous-averaging Asian option.*

- class [DiscreteAveragingAsianOption](#)

*Discrete-averaging Asian option.*

- class [AssetSwap](#)

*Asset swap.*

- class [BarrierOption](#)

*Barrier option on a single asset.*

- class [BasketOption](#)

*Basket option on a number of assets.*

- class [Bond](#)

*Base bond class.*

- class [CapFloor](#)

*Base class for cap-like instruments.*

- class [Cap](#)

*Concrete cap class.*

- class [Floor](#)

*Concrete floor class.*

- class [Collar](#)

*Concrete collar class.*

- class [CliquetOption](#)

*cliquet (Ratchet) option*

- class [CompositeInstrument](#)

*Composite instrument.*

- class [DividendVanillaOption](#)  
*Single-asset vanilla option (no barriers) with discrete dividends.*
- class [EuropeanOption](#)  
*European option on a single asset.*
- class [FixedCouponBond](#)  
*fixed-coupon bond*
- class [FixedCouponBondForward](#)  
*forward contract on a fixed-coupon bond*
- class [FloatingRateBond](#)  
*floating-rate bond*
- class [Forward](#)  
*Abstract base forward class.*
- class [ForwardRateAgreement](#)  
*Forward rate agreement (FRA) class.*
- class [ForwardVanillaOption](#)  
*Forward version of a vanilla option.*
- class [ContinuousFloatingLookbackOption](#)  
*Continuous-floating lookback option.*
- class [ContinuousFixedLookbackOption](#)  
*Continuous-fixed lookback option.*
- class [QuantoForwardVanillaOption](#)  
*Quanto version of a forward vanilla option.*
- class [QuantoVanillaOption](#)  
*quanto version of a vanilla option*
- class [Stock](#)  
*Simple stock class.*
- class [Swap](#)  
*Interest rate swap.*
- class [Swaption](#)  
*Swaption class*
- class [VanillaOption](#)  
*Vanilla option (no discrete dividends, no barriers) on a single asset.*
- class [VanillaSwap](#)  
*Plain-vanilla swap.*

- class [VarianceSwap](#)  
*Variance swap.*
- class [ZeroCouponBond](#)  
*zero-coupon bond*

## 6.19 Lattice methods

### 6.19.1 Detailed Description

The framework (corresponding to the `ql/Lattices` directory) contains basic building blocks for pricing instruments using lattice methods (trees). A lattice, i.e. an instance of the abstract class `QuantLib::Lattice`, 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::Tree`, classes which define the branching between nodes and transition probabilities.

### 6.19.2 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 `bsmllattice.hpp` 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.

### 6.19.3 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::Trinomial-Tree` 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

$$p_u y_{i+1,k+1}^2 + p_m y_{i+1,k}^2 + p_d y_{i+1,k-1}^2 = V_{i,j}^2 + E_{i,j}^2,$$

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,  $E_{i,j} = \mathbf{E}(y(t_{i+1})|y(t_i) = y_{i,j})$  and  $V_{i,j}^2 = \mathbf{Var}\{y(t_{i+1})|y(t_i) = y_{i,j}\}$ .

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

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

### 6.19.4 Bidimensional lattices

To come...

### 6.19.5 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](#), 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::DiscretizedSwap](#) and [QuantLib::DiscretizedSwaption](#).

#### Classes

- class [BinomialTree](#)  
*Binomial tree base class.*
- class [EqualProbabilitiesBinomialTree](#)  
*Base class for equal probabilities binomial tree.*
- class [EqualJumpsBinomialTree](#)  
*Base class for equal jumps binomial tree.*
- class [JarrowRudd](#)  
*Jarrow-Rudd (multiplicative) equal probabilities binomial tree.*
- class [CoxRossRubinstein](#)  
*Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.*
- class [AdditiveEQPBinomialTree](#)  
*Additive equal probabilities binomial tree.*
- class [Trigeorgis](#)  
*Trigeorgis (additive equal jumps) binomial tree*
- class [Tian](#)  
*Tian tree: third moment matching, multiplicative approach*
- class [LeisenReimer](#)  
*Leisen & Reimer tree: multiplicative approach.*
- class [BlackScholesLattice](#)  
*Simple binomial lattice approximating the Black-Scholes model.*
- class [Lattice](#)  
*Lattice-method base class.*
- class [Lattice1D](#)

*One-dimensional lattice.*

- class [Lattice2D](#)

*Two-dimensional lattice.*

- class [TsiveriotisFernandesLattice](#)

*Binomial lattice approximating the Tsiveriotis-Fernandes model.*

- class [Tree](#)

*Tree approximating a single-factor diffusion*

- class [TrinomialTree](#)

*Recombining trinomial tree class.*

## 6.20 Math tools

Math facilities of the library include:

### 6.20.1 Pseudo-random number and low-discrepancy sequence generators

Implementations of pseudo-random number and low-discrepancy sequence generators. They share the `ql/RandomNumbers` directory.

### 6.20.2 One-dimensional solvers

The abstract class [QuantLib::Solver1D](#) 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 `ql/Solvers1D` directory.

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 definition: for some algorithms it is the  $x$ -accuracy, for others it is  $f(x)$ -accuracy.

### 6.20.3 Optimizers

The optimization framework (corresponding to the `ql/Optimization` directory) implements some multi-dimensional minimizing methods. The function to be minimized is to be derived from the [QuantLib::CostFunction](#) base class (if the gradient is not analytically implemented, it will be computed numerically).

#### The simplex method

This method, implemented in [QuantLib::Simplex](#), 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  $\lambda$  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::ConjugateGradient](#). At each step, we minimize (using Armijo's line search algorithm, implemented in [QuantLib::ArmijoLineSearch](#)) the function along a line defined by

$$\mathbf{d}_i = -\nabla f(\mathbf{x}_i) + \frac{\|\nabla f(\mathbf{x}_i)\|^2}{\|\nabla f(\mathbf{x}_{i-1})\|^2} \mathbf{d}_{i-1},$$



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

As we can see, this optimization method requires the knowledge of the gradient of the cost function. See [QuantLib::ConjugateGradient](#).

## 6.21 Monte Carlo framework

### 6.21.1 Detailed Description

This framework (corresponding to the ql/MonteCarlo directory) contains basic building blocks for Monte Carlo simulations.

#### Classes

- class [BrownianBridge](#)  
*Builds Wiener process paths using Gaussian variates.*
- class [EarlyExercisePathPricer](#)  
*base class for early exercise path pricers*
- class [LongstaffSchwartzPathPricer](#)  
*Longstaff-Schwarz path pricer for early exercise options.*
- class [MonteCarloModel](#)  
*General purpose Monte Carlo model for path samples.*
- class [MultiPath](#)  
*Correlated multiple asset paths.*
- class [MultiPathGenerator](#)  
*Generates a multipath from a random number generator.*
- class [Path](#)
- class [PathGenerator](#)  
*Generates random paths using a sequence generator.*
- class [PathPricer](#)  
*base class for path pricers*
- struct [Sample](#)  
*weighted sample*

## 6.22 Design patterns

### Classes

- class [Bridge](#)  
*The Bridge pattern made explicit.*
- class [Composite](#)  
*Composite pattern.*
- 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 [Singleton](#)  
*Basic support for the singleton pattern.*
- class [AcyclicVisitor](#)  
*degenerate base class for the Acyclic Visitor pattern*

## 6.23 Stochastic processes

### 6.23.1 Detailed Description

The classes [QuantLib::StochasticProcess](#) and [QuantLib::StochasticProcess1D](#) provide the interface for a generic stochastic process. A number of specific processes is contained in the `ql/Processes` directory.

#### Classes

- class [GeneralizedBlackScholesProcess](#)  
*Generalized Black-Scholes stochastic process.*
- class [BlackScholesProcess](#)  
*Black-Scholes (1973) stochastic process.*
- class [BlackScholesMertonProcess](#)  
*Merton (1973) extension to the Black-Scholes stochastic process.*
- class [BlackProcess](#)  
*Black (1976) stochastic process.*
- class [GarmanKohlagenProcess](#)  
*Garman-Kohlhagen (1983) stochastic process.*
- class [EulerDiscretization](#)  
*Euler discretization for stochastic processes.*
- class [ForwardMeasureProcess](#)  
*forward-measure stochastic process*
- class [ForwardMeasureProcess1D](#)  
*forward-measure 1-D stochastic process*
- class [G2Process](#)  
*G2 stochastic process*
- class [G2ForwardProcess](#)  
*forward G2 stochastic process*
- class [GeometricBrownianMotionProcess](#)  
*Geometric brownian-motion process.*
- class [HestonProcess](#)  
*Square-root stochastic-volatility Heston process.*
- class [HullWhiteProcess](#)  
*Hull-White stochastic process.*

- class [HullWhiteForwardProcess](#)  
*forward Hull-White stochastic process*
- class [LiborForwardModelProcess](#)  
*libor-forward-model process*
- class [Merton76Process](#)  
*Merton-76 jump-diffusion process.*
- class [OrnsteinUhlenbeckProcess](#)  
*Ornstein-Uhlenbeck process class.*
- class [SquareRootProcess](#)  
*Square-root process class.*
- class [StochasticProcessArray](#)  
*Array of correlated 1-D stochastic processes*

## 6.24 Term structures

### 6.24.1 Detailed Description

The abstract class [QuantLib::YieldTermStructure](#) provides the common interface to concrete yield-rate 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.

#### Classes

- class [InterpolatedDiscountCurve](#)  
*Term structure based on interpolation of discount factors.*
- class [FlatForward](#)  
*Flat interest-rate curve.*
- class [InterpolatedForwardCurve](#)  
*Term structure based on interpolation of forward rates.*
- class [ForwardSpreadedTermStructure](#)  
*Term structure with added spread on the instantaneous forward rate.*
- class [ForwardRateStructure](#)  
*Forward rate term structure.*
- class [ImpliedTermStructure](#)  
*Implied term structure at a given date in the future.*
- class [PiecewiseYieldCurve](#)  
*Piecewise yield term structure.*
- class [PiecewiseZeroSpreadedTermStructure](#)  
*Term structure with an added vector of spreads on the zero-yield rate.*
- class [InterpolatedZeroCurve](#)  
*Term structure based on interpolation of zero yields.*
- class [ZeroSpreadedTermStructure](#)  
*Term structure with an added spread on the zero yield rate.*
- class [ZeroYieldStructure](#)  
*Zero-yield term structure.*
- class [YieldTermStructure](#)  
*Interest-rate term structure.*

## Typedefs

- `typedef InterpolatedDiscountCurve< LogLinear > DiscountCurve`  
*Term structure based on log-linear interpolation of discount factors.*
- `typedef InterpolatedForwardCurve< BackwardFlat > ForwardCurve`  
*Term structure based on flat interpolation of forward rates.*
- `typedef PiecewiseYieldCurve< Discount, LogLinear > PiecewiseFlatForward`  
*Piecewise flat-forward term structure.*
- `typedef InterpolatedZeroCurve< Linear > ZeroCurve`  
*Term structure based on linear interpolation of zero yields.*

### 6.24.2 Typedef Documentation

#### 6.24.2.1 `typedef InterpolatedDiscountCurve<LogLinear> DiscountCurve`

Term structure based on log-linear interpolation of discount factors.

Log-linear interpolation guarantees piecewise-constant forward rates.

## 6.25 Utilities

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`, 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::coupling_iterator` 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  $[x_0, x_1, \dots]$  and a predicate  $p$  and generates the sequence of those  $x_i$  for which  $p(x_i)$  returns `true`;



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

## 6.26 QuantLib macros

### 6.26.1 Detailed Description

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

#### Modules

- [Generic macros](#)
- [Numeric limits](#)
- [Template capabilities](#)
- [Iterator support](#)
- [Debugging macros](#)

#### Defines

- `#define QL\_VERSION "0.3.14"`  
*version string*
- `#define QL\_HEX\_VERSION 0x000314f0`  
*version hexadecimal number*
- `#define QL\_LIB\_VERSION "0_3_14"`  
*version string for output lib name*

## 6.27 Generic macros

### 6.27.1 Detailed Description

Miscellaneous macros for compiler idiosyncrasies not fitting other categories.

#### Defines

- `#define QL_DUMMY_RETURN(x)`  
*Is a dummy return statement required?*
- `#define QL_IO_INIT`  
*I/O initialization.*

### 6.27.2 Define Documentation

#### 6.27.2.1 `#define QL_DUMMY_RETURN(x)`

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
    QL_FAIL("whatever the reason");
```

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.

#### 6.27.2.2 `#define QL_IO_INIT`

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.

#### Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), [FRA.cpp](#), [Replication.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

## 6.28 Numeric limits

### 6.28.1 Detailed Description

Some compilers do not give an implementation of `<limits>` 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_INTEGER ((std::numeric_limits<QL_INTEGER>::min)())`
- `#define QL_MAX_INTEGER ((std::numeric_limits<QL_INTEGER>::max)())`
- `#define QL_MIN_REAL -((std::numeric_limits<QL_REAL>::max)())`
- `#define QL_MIN_POSITIVE_REAL ((std::numeric_limits<QL_REAL>::min)())`
- `#define QL_MAX_REAL ((std::numeric_limits<QL_REAL>::max)())`
- `#define QL_EPSILON ((std::numeric_limits<QL_REAL>::epsilon)())`
- `#define QL_NULL_INTEGER ((std::numeric_limits<int>::max)())`
- `#define QL_NULL_REAL ((std::numeric_limits<float>::max)())`

### 6.28.2 Define Documentation

#### 6.28.2.1 `#define QL_MIN_INTEGER ((std::numeric_limits<QL_INTEGER>::min)())`

Defines the value of the largest representable negative integer value

#### 6.28.2.2 `#define QL_MAX_INTEGER ((std::numeric_limits<QL_INTEGER>::max)())`

Defines the value of the largest representable integer value

#### 6.28.2.3 `#define QL_MIN_REAL -((std::numeric_limits<QL_REAL>::max)())`

Defines the value of the largest representable negative floating-point value

#### 6.28.2.4 `#define QL_MIN_POSITIVE_REAL ((std::numeric_limits<QL_REAL>::min)())`

Defines the value of the smallest representable positive double value

#### 6.28.2.5 `#define QL_MAX_REAL ((std::numeric_limits<QL_REAL>::max)())`

Defines the value of the largest representable floating-point value

#### 6.28.2.6 `#define QL_EPSILON ((std::numeric_limits<QL_REAL>::epsilon)())`

Defines the machine precision for operations over doubles

## 6.29 Template capabilities

### 6.29.1 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_TYPENAME typename`

### 6.29.2 Define Documentation

#### 6.29.2.1 `#define QL_TYPENAME typename`

In Visual C++ 6, `typename` can only be used in template declarations and not in template definitions.

## 6.30 Iterator support

### 6.30.1 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\_FULL\_ITERATOR\_SUPPORT`

### 6.30.2 Define Documentation

#### 6.30.2.1 `#define QL_FULL_ITERATOR_SUPPORT`

Some compilers (most notably, Visual C++ 6) 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.

## 6.31 Output manipulators

### 6.31.1 Detailed Description

Helper functions for creating formatted output.

#### Functions

- detail::long\_weekday\_holder [long\\_weekday](#) (Weekday)  
*output weekdays in long format*
- detail::short\_weekday\_holder [short\\_weekday](#) (Weekday)  
*output weekdays in short format (three letters)*
- detail::shortest\_weekday\_holder [shortest\\_weekday](#) (Weekday)  
*output weekdays in shortest format (two letters)*
- detail::short\_date\_holder [short\\_date](#) (const Date &)  
*output dates in short format (mm/dd/yyyy)*
- detail::long\_date\_holder [long\\_date](#) (const Date &)  
*output dates in long format (Month ddth, yyyy)*
- detail::iso\_date\_holder [iso\\_date](#) (const Date &)  
*output dates in ISO format (yyyy-mm-dd)*
- detail::long\_period\_holder [long\\_period](#) (const Period &)  
*output periods in long format (e.g. "2 weeks")*
- detail::short\_period\_holder [short\\_period](#) (const Period &)  
*output periods in short format (e.g. "2w")*
- template<typename T> detail::null\_checker< T > [checknull](#) (T)  
*check for nulls before output*
- detail::ordinal\_holder [ordinal](#) (Size)  
*outputs naturals as 1st, 2nd, 3rd...*
- template<typename T> detail::power\_of\_two\_holder< T > [power\\_of\\_two](#) (T)  
*output integers as powers of two*
- detail::percent\_holder [percent](#) (Real)  
*output reals as percentages*
- detail::percent\_holder [rate](#) (Rate)  
*output rates and spreads as percentages*
- detail::percent\_holder [volatility](#) (Volatility)  
*output volatilities as percentages*

## 6.32 Debugging macros

### 6.32.1 Detailed Description

For debugging purposes, macros can be used to output information about the code being executed.

#### Defines

- `#define QL_TRACE_ENABLE`  
*enable tracing*
- `#define QL_TRACE_DISABLE`  
*disable tracing*
- `#define QL_TRACE_ON(out)`  
*set tracing stream*
- `#define QL_TRACE(message)`  
*output tracing information*
- `#define QL_TRACE_ENTER_FUNCTION`  
*output tracing information*
- `#define QL_TRACE_EXIT_FUNCTION`  
*output tracing information*
- `#define QL_TRACE_LOCATION`  
*output tracing information*
- `#define QL_TRACE_VARIABLE(variable)`  
*output tracing information*

### 6.32.2 Define Documentation

#### 6.32.2.1 `#define QL_TRACE_ENABLE`

enable tracing

The statement

```
QL_TRACE_ENABLE;
```

can be used to enable tracing. Such statement might be ignored; refer to `QL_TRACE` for details.

#### Examples:

[tracing\\_example.cpp](#).



### 6.32.2.2 **#define QL\_TRACE\_DISABLE**

disable tracing

The statement

```
QL_TRACE_DISABLE;
```

can be used to disable tracing. Such statement might be ignored; refer to QL\_TRACE for details.

### 6.32.2.3 **#define QL\_TRACE\_ON(out)**

set tracing stream

The statement

```
QL_TRACE_ON(stream);
```

can be used to set the stream where tracing messages are output. Such statement might be ignored; refer to QL\_TRACE for details.

### 6.32.2.4 **#define QL\_TRACE(message)**

output tracing information

The statement

```
QL_TRACE(message);
```

can be used to output a trace of the code being executed. If tracing was disabled during configuration, such statements are removed by the preprocessor for maximum performance; if it was enabled, whether and where the message is output depends on the current settings.

**Examples:**

[tracing\\_example.cpp](#).

### 6.32.2.5 **#define QL\_TRACE\_ENTER\_FUNCTION**

output tracing information

The statement

```
QL_TRACE_ENTER_FUNCTION;
```

can be used at the beginning of a function to trace the fact that the program execution is entering such function. It should be paired with a corresponding QL\_TRACE\_EXIT\_FUNCTION macro. Such statement might be ignored; refer to QL\_TRACE for details. Also, function information might not be available depending on the compiler.

**Examples:**

[tracing\\_example.cpp](#).

#### 6.32.2.6 **#define QL\_TRACE\_EXIT\_FUNCTION**

output tracing information

The statement

```
QL_TRACE_EXIT_FUNCTION;
```

can be used before returning from a function to trace the fact that the program execution is exiting such function. It should be paired with a corresponding `QL_TRACE_ENTER_FUNCTION` macro. Such statement might be ignored; refer to `QL_TRACE` for details. Also, function information might not be available depending on the compiler.

**Examples:**

[tracing\\_example.cpp](#).

#### 6.32.2.7 **#define QL\_TRACE\_LOCATION**

output tracing information

The statement

```
QL_TRACE_LOCATION;
```

can be used to trace the current file and line. Such statement might be ignored; refer to `QL_TRACE` for details.

**Examples:**

[tracing\\_example.cpp](#).

#### 6.32.2.8 **#define QL\_TRACE\_VARIABLE(variable)**

output tracing information

The statement

```
QL_TRACE_VARIABLE(variable);
```

can be used to trace the current value of a variable. Such statement might be ignored; refer to `QL_TRACE` for details. Also, the variable type must allow sending it to an output stream.

**Examples:**

[tracing\\_example.cpp](#).

# Chapter 7

## QuantLib Class Documentation

### 7.1 Abcd Class Reference

```
#include <ql/MarketModels/Models/abcd.hpp>
```

#### 7.1.1 Detailed Description

[Abcd](#) functional form for instantaneous volatility.

$$f(T - t) = [a + b(T - t)]e^{-c(T-t)} + d$$

following Rebonato notation.

#### Public Member Functions

- **Abcd** (Real a=-0.06, Real b=0.17, Real c=0.54, Real d=0.17, bool aIsFixed=false, bool bIsFixed=false, bool cIsFixed=false, bool dIsFixed=false)
- Real [operator\(\)](#) (Time u) const

*instantaneous volatility at time to maturity u:*

$$f(u)$$

- Real **a** () const
- Real **b** () const
- Real **c** () const
- Real **d** () const
- Real [instantaneousVolatility](#) (Time u, Time T) const
- Real [instantaneousVariance](#) (Time u, Time T) const
- Real [instantaneousCovariance](#) (Time u, Time T, Time S) const
- Real [volatility](#) (Time tMin, Time tMax, Time T) const
- Real [variance](#) (Time tMin, Time tMax, Time T) const
- Real [covariance](#) (Time tMin, Time tMax, Time T, Time S) const
- Real [shortTermVolatility](#) () const

*instantaneous volatility when time to maturity = 0.0*

- Real [longTermVolatility](#) () const  
*instantaneous volatility when time to maturity = +inf*
- Real [maximumLocation](#) () const  
*time to maturity at which the instantaneous volatility reaches maximum (if any)*
- Real [maximumVolatility](#) () const  
*maximum of the instantaneous volatility*
- std::vector< Real > [k](#) (const std::vector< Real > &blackVols, const std::vector< Real >::const\_iterator &t) const  
*adjustment factors needed to match Black vols*
- Real [error](#) (const std::vector< Real > &blackVols, const std::vector< Real >::const\_iterator &t) const  
*vol error*
- EndCriteria::Type [capletCalibration](#) (const std::vector< Real > &blackVols, const std::vector< Real >::const\_iterator &t, const boost::shared\_ptr< [OptimizationMethod](#) > &method=boost::shared\_ptr< [OptimizationMethod](#) >())  
*calibration*

## Friends

- class [AbcdCostFunction](#)

## 7.1.2 Member Function Documentation

### 7.1.2.1 Real instantaneousVolatility (Time $u$ , Time $T$ ) const

instantaneous volatility at time  $u$  of the  $T$ -fixing rate:

$$f(T - u)$$

### 7.1.2.2 Real instantaneousVariance (Time $u$ , Time $T$ ) const

instantaneous variance at time  $u$  of  $T$ -fixing rate:

$$f(T - u)f(T - u)$$

### 7.1.2.3 Real instantaneousCovariance (Time $u$ , Time $T$ , Time $S$ ) const

instantaneous covariance at time  $u$  between  $T$  and  $S$  fixing rates:

$$f(T - u)f(S - u)$$

**7.1.2.4 Real volatility (Time  $tMin$ , Time  $tMax$ , Time  $T$ ) const**

volatility in  $[tMin, tMax]$  of T-fixing rate:

$$\sqrt{\int_{tMin}^{tMax} f^2(T - u) du}$$

**7.1.2.5 Real variance (Time  $tMin$ , Time  $tMax$ , Time  $T$ ) const**

variance in  $[tMin, tMax]$  of T-fixing rate:

$$\int_{tMin}^{tMax} f^2(T - u) du$$

**7.1.2.6 Real covariance (Time  $tMin$ , Time  $tMax$ , Time  $T$ , Time  $S$ ) const**

covariance in  $[tMin, tMax]$  between T and S fixing rates:

$$\int_{tMin}^{tMax} f(T - u) f(S - u) du$$

## 7.2 AbcdSquared Class Reference

```
#include <ql/MarketModels/Models/abcd.hpp>
```

### 7.2.1 Detailed Description

[Abcd](#) Squared functional. Helper class.

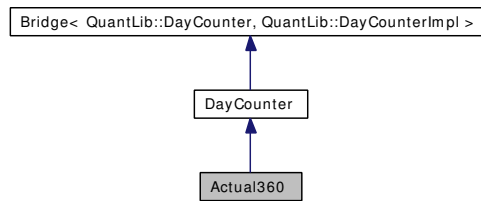
#### Public Member Functions

- **AbcdSquared** (Real a, Real b, Real c, Real d, Time S, Time T)
- Real **operator()** (Time u) const

## 7.3 Actual360 Class Reference

```
#include <ql/DayCounters/actual360.hpp>
```

Inheritance diagram for Actual360:



### 7.3.1 Detailed Description

Actual/360 day count convention.

Actual/360 day count convention, also known as "Act/360", or "A/360".

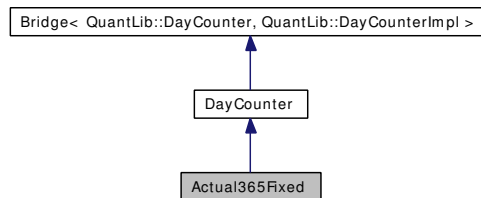
**Examples:**

[Repo.cpp](#), and [swapvaluation.cpp](#).

## 7.4 Actual365Fixed Class Reference

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Inheritance diagram for Actual365Fixed:



### 7.4.1 Detailed Description

Actual/365 (Fixed) day count convention.

"Actual/365 (Fixed)" day count convention, also known as "Act/365 (Fixed)", "A/365 (Fixed)", or "A/365F".

#### Warning

According to ISDA, "Actual/365" (without "Fixed") is an alias for "Actual/Actual (ISDA)" (see [ActualActual](#)). If Actual/365 is not explicitly specified as fixed in an instrument specification, you might want to double-check its meaning.

#### Examples:

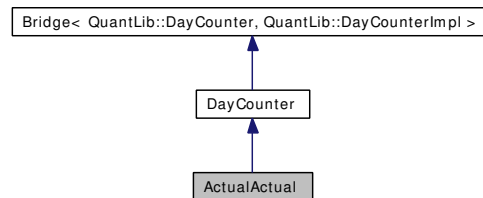
[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), and [Replication.cpp](#).



## 7.5 ActualActual Class Reference

```
#include <ql/DayCounters/actualactual.hpp>
```

Inheritance diagram for ActualActual:



### 7.5.1 Detailed Description

Actual/Actual day count.

The day count can be calculated according to:

- the ISDA convention, also known as "Actual/Actual (Historical)", "Actual/Actual", "Act/Act", and according to ISDA also "Actual/365", "Act/365", and "A/365";
- the ISMA and US Treasury convention, also known as "Actual/Actual (Bond)";
- the AFB convention, also known as "Actual/Actual (Euro)".

For more details, refer to <http://www.isda.org/publications/pdf/Day-Count-Fraction1999.pdf>

#### Tests

the correctness of the results is checked against known good values.

#### Examples:

[FRA.cpp](#), and [swapvaluation.cpp](#).

### Public Types

- enum **Convention** {  
ISMA, Bond, ISDA, Historical,  
Actual365, AFB, Euro }

### Public Member Functions

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

## 7.6 AcyclicVisitor Class Reference

```
#include <ql/Patterns/visitor.hpp>
```

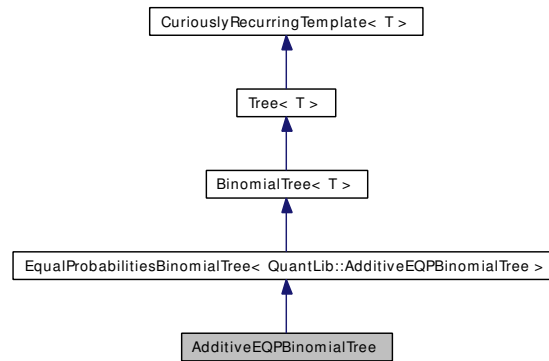
### 7.6.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

## 7.7 AdditiveEQPBinoialTree Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for AdditiveEQPBinoialTree:



### 7.7.1 Detailed Description

Additive equal probabilities binomial tree.

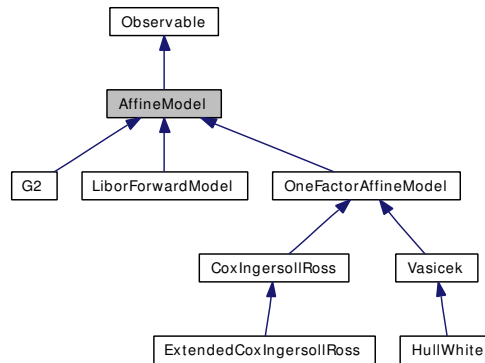
#### Public Member Functions

- **AdditiveEQPBinoialTree** (const boost::shared\_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)

## 7.8 AffineModel Class Reference

```
#include <ql/ShortRateModels/model.hpp>
```

Inheritance diagram for AffineModel:



### 7.8.1 Detailed Description

Affine model class.

Base class for analytically tractable models.

#### Public Member Functions

- virtual [DiscountFactor](#) **discount** (Time t) const=0  
*Implied discount curve.*
- virtual Real **discountBond** (Time now, Time maturity, [Array](#) factors) const=0
- virtual Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const =0

## 7.9 AmericanCondition Class Reference

```
#include <ql/FiniteDifferences/americancondition.hpp>
```

### 7.9.1 Detailed Description

American exercise condition.

#### Todo

unify the intrinsicValues/Payoff thing

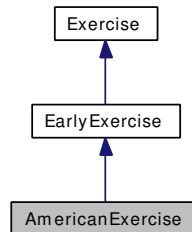
### Public Member Functions

- **AmericanCondition** (Option::Type type, Real strike)
- **AmericanCondition** (const [Array](#) &intrinsicValues)

## 7.10 AmericanExercise Class Reference

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

Inheritance diagram for AmericanExercise:



### 7.10.1 Detailed Description

American exercise.

An American option can be exercised at any time between two predefined dates; the first date might be omitted, in which case the option can be exercised at any time before the expiry.

#### Todo

check that everywhere the American condition is applied from earliestDate and not earlier

#### Examples:

[ConvertibleBonds.cpp](#), and [EquityOption.cpp](#).

### Public Member Functions

- **AmericanExercise** (const [Date](#) &earliestDate, const [Date](#) &latestDate, bool payoffAtExpiry=false)
- **AmericanExercise** (const [Date](#) &latestDate, bool payoffAtExpiry=false)

## 7.11 AmericanPayoffAtExpiry Class Reference

```
#include <ql/PricingEngines/americanpayoffatexpiry.hpp>
```

### 7.11.1 Detailed Description

Analytic formula for American exercise payoff at-expiry options

#### Todo

calculate greeks

### Public Member Functions

- **AmericanPayoffAtExpiry** (Real spot, DiscountFactor discount, DiscountFactor dividendDiscount, Real variance, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff)
- Real **value** () const

## 7.12 AmericanPayoffAtHit Class Reference

```
#include <ql/PricingEngines/americanpayoffathit.hpp>
```

### 7.12.1 Detailed Description

Analytic formula for American exercise payoff at-hit options

#### Todo

calculate greeks

### Public Member Functions

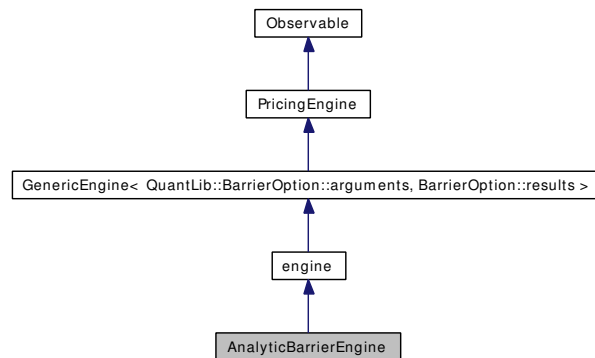
- **AmericanPayoffAtHit** (Real spot, DiscountFactor discount, DiscountFactor dividendDiscount, Real variance, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff)
- Real **value** () const
- Real **delta** () const
- Real **gamma** () const
- Real **rho** ([Time](#) maturity) const



## 7.13 AnalyticBarrierEngine Class Reference

```
#include <ql/PricingEngines/Barrier/analyticbarrierengine.hpp>
```

Inheritance diagram for AnalyticBarrierEngine:



### 7.13.1 Detailed Description

Pricing engine for barrier options using analytical formulae.

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

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

#### Todo

rework to avoid repeated casts inside utility methods

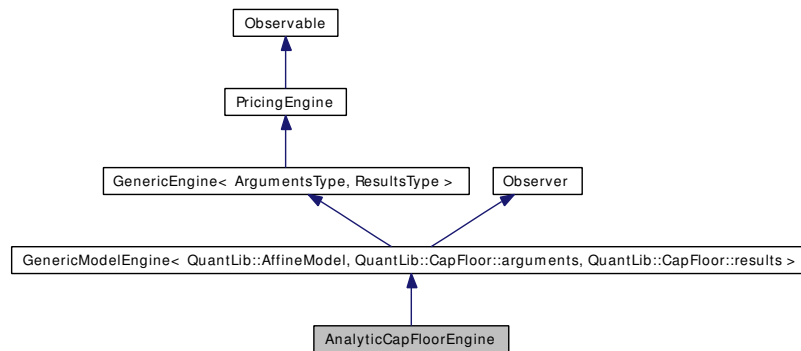
### Public Member Functions

- void **calculate** () const

## 7.14 AnalyticCapFloorEngine Class Reference

```
#include <ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp>
```

Inheritance diagram for AnalyticCapFloorEngine:



### 7.14.1 Detailed Description

Analytic engine for cap/floor.

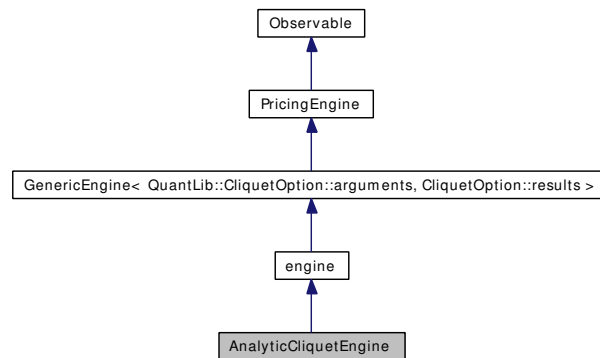
#### Public Member Functions

- **AnalyticCapFloorEngine** (const boost::shared\_ptr< [AffineModel](#) > &model)
- void **calculate** () const

## 7.15 AnalyticCliquetEngine Class Reference

```
#include <ql/PricingEngines/Cliquet/analyticcliquetengine.hpp>
```

Inheritance diagram for AnalyticCliquetEngine:



### 7.15.1 Detailed Description

Pricing engine for Cliquet options using analytical formulae.

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

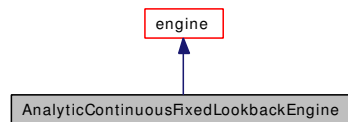
### Public Member Functions

- void **calculate** () const

## 7.16 AnalyticContinuousFixedLookbackEngine Class Reference

```
#include <ql/PricingEngines/Lookback/analyticcontinuousfixedlookback.hpp>
```

Inheritance diagram for AnalyticContinuousFixedLookbackEngine:



### 7.16.1 Detailed Description

Pricing engine for European continuous fixed-strike lookback.

Formula from "Option Pricing Formulas", E.G. Haug, McGraw-Hill, 1998, p.63-64

#### Tests

returned values are verified against results from literature

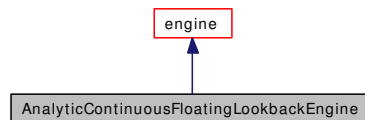
### Public Member Functions

- void `calculate()` const

## 7.17 AnalyticContinuousFloatingLookbackEngine Class Reference

```
#include <ql/PricingEngines/Lookback/analyticcontinuousfloatinglookback.hpp>
```

Inheritance diagram for AnalyticContinuousFloatingLookbackEngine:



### 7.17.1 Detailed Description

Pricing engine for European continuous floating-strike lookback.

Formula from "Option Pricing Formulas", E.G. Haug, McGraw-Hill, 1998, p.61-62

#### Tests

returned values verified against results from literature

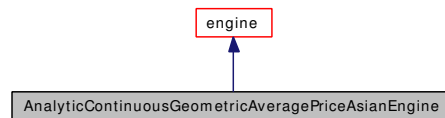
### Public Member Functions

- void **calculate** () const

## 7.18 AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference

```
#include <ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp>
```

Inheritance diagram for AnalyticContinuousGeometricAveragePriceAsianEngine:



### 7.18.1 Detailed Description

Pricing engine for European continuous geometric average price Asian.

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.

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature, and results obtained using a discrete average approximation.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

#### Todo

handle seasoned options

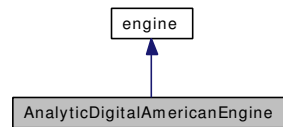
### Public Member Functions

- void **calculate** () const

## 7.19 AnalyticDigitalAmericanEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp>
```

Inheritance diagram for AnalyticDigitalAmericanEngine:



### 7.19.1 Detailed Description

Pricing engine for American vanilla options with digital payoff using analytic formulae

#### Todo

add more greeks (as of now only delta and rho available)

#### Tests

- the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of cash-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
- the correctness of the returned greeks in case of cash-or-nothing at-hit digital payoff is tested by reproducing numerical derivatives.

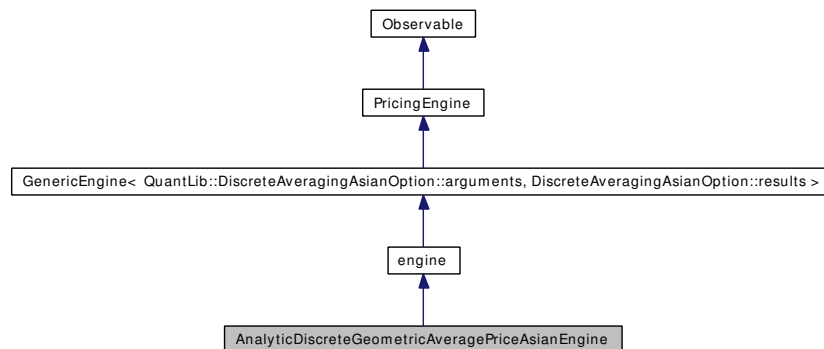
### Public Member Functions

- void `calculate ()` const

## 7.20 AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference

```
#include <ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp>
```

Inheritance diagram for AnalyticDiscreteGeometricAveragePriceAsianEngine:



### 7.20.1 Detailed Description

Pricing engine for European discrete geometric average price Asian.

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, pag 65-97

#### Todo

implement correct theta, rho, and dividend-rho calculation

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the available greeks is tested against numerical calculations.

### Public Member Functions

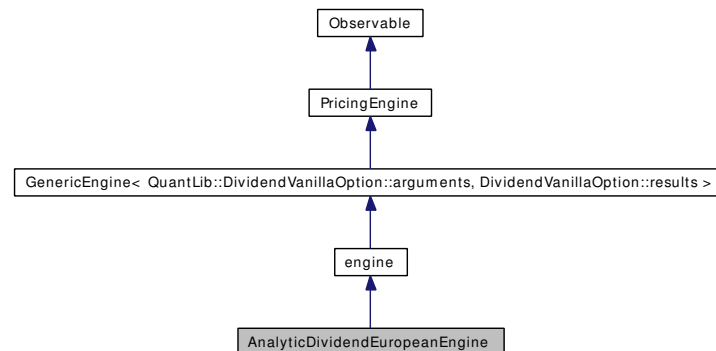
- void **calculate** () const



## 7.21 AnalyticDividendEuropeanEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/analyticdividend europeanengine.hpp>
```

Inheritance diagram for AnalyticDividendEuropeanEngine:



### 7.21.1 Detailed Description

Analytic pricing engine for European options with discrete dividends.

#### Tests

the correctness of the returned greeks is tested by reproducing numerical derivatives.

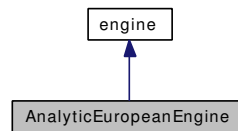
### Public Member Functions

- `void calculate () const`

## 7.22 AnalyticEuropeanEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp>
```

Inheritance diagram for AnalyticEuropeanEngine:



### 7.22.1 Detailed Description

Pricing engine for European vanilla options using analytical formulae.

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the implied-volatility calculation is tested by checking that it does not modify the option.
- the correctness of the returned value in case of cash-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of gap digital payoff is tested by reproducing results available in literature.
- the correctness of the returned greeks in case of cash-or-nothing digital payoff is tested by reproducing numerical derivatives.

Examples:

[EquityOption.cpp](#).

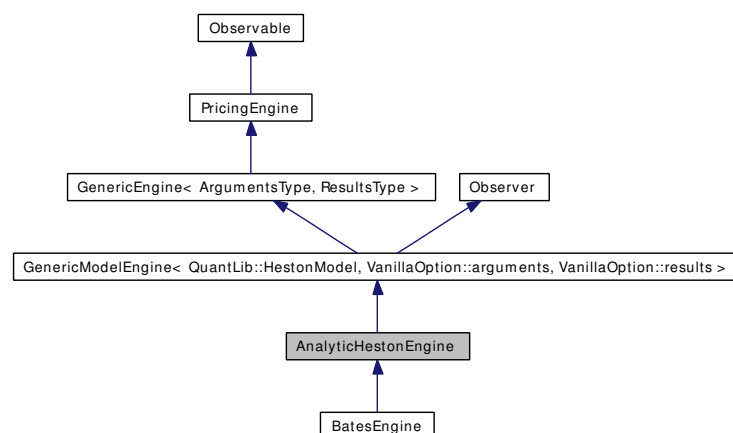
### Public Member Functions

- void **calculate** () const

## 7.23 AnalyticHestonEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/analytichestonengine.hpp>
```

Inheritance diagram for AnalyticHestonEngine:



### 7.23.1 Detailed Description

analytic Heston-model engine based on Fourier transform

References:

Heston, Steven L., 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to [Bond](#) and [Currency](#) Options. The review of Financial Studies, Volume 6, Issue 2, 327-343.

Dupire, Bruno, 1994. Pricing with a smile. Risk Magazine, 7, 18-20.

A. Sepp, Pricing European-Style Options under Jump Diffusion Processes with Stochastic Volatility: Applications of Fourier Transform (<http://math.ut.ee/~spartak/papers/stochjumpvols.pdf>)

#### Tests

the correctness of the returned value is tested by reproducing results available in web/literature and comparison with Black pricing.

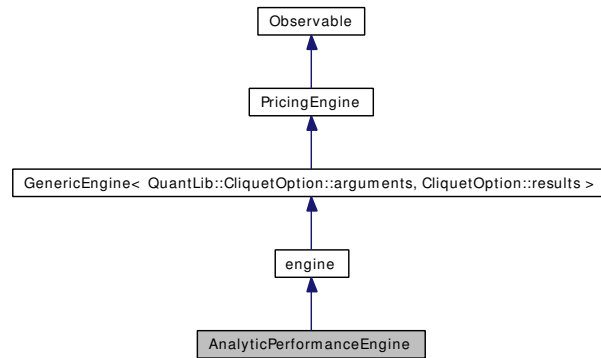
### Public Member Functions

- **AnalyticHestonEngine** (const boost::shared\_ptr< [HestonModel](#) > &model, Size integrationOrder=64)
- void **calculate** () const
- virtual std::complex< Real > **jumpDiffusionTerm** (Real phi, Time t, Size j) const

## 7.24 AnalyticPerformanceEngine Class Reference

```
#include <ql/PricingEngines/Cliquet/analyticperformanceengine.hpp>
```

Inheritance diagram for AnalyticPerformanceEngine:



### 7.24.1 Detailed Description

Pricing engine for performance options using analytical formulae.

#### Tests

the correctness of the returned greeks is tested by reproducing numerical derivatives.

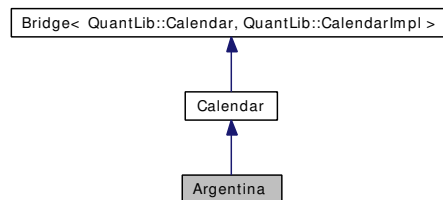
### Public Member Functions

- void **calculate** () const

## 7.25 Argentina Class Reference

```
#include <ql/Calendars/argentina.hpp>
```

Inheritance diagram for Argentina:



### 7.25.1 Detailed Description

Argentinian calendars.

Holidays for the Buenos Aires stock exchange (data from <http://www.merval.sba.com.ar/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Holy Thursday
- Good Friday
- Labour Day, May 1st
- May Revolution, May 25th
- Death of General Manuel Belgrano, third Monday of June
- Independence Day, July 9th
- Death of General José de San Martín, third Monday of August
- Columbus Day, October 12th (moved to preceding Monday if on Tuesday or Wednesday and to following if on Thursday or Friday)
- Immaculate Conception, December 8th
- Christmas Eve, December 24th
- New Year's Eve, December 31th

### Public Types

- enum [Market](#) { [Merval](#) }

### Public Member Functions

- [Argentina](#) ([Market](#) m=Merval)

## 7.25.2 Member Enumeration Documentation

### 7.25.2.1 enum [Market](#)

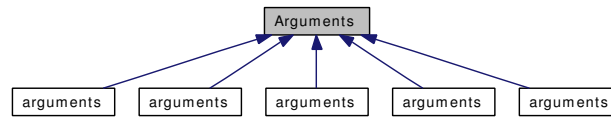
Enumerator:

*Merval* Buenos Aires stock exchange calendar.

## 7.26 Arguments Class Reference

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

Inheritance diagram for Arguments:



### 7.26.1 Detailed Description

base class for generic argument groups

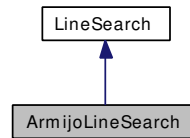
#### Public Member Functions

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

## 7.27 ArmijoLineSearch Class Reference

```
#include <ql/Optimization/armijo.hpp>
```

Inheritance diagram for ArmijoLineSearch:



### 7.27.1 Detailed Description

Armijo line search.

Let  $\alpha$  and  $\beta$  be 2 scalars in  $[0, 1]$ . Let  $x$  be the current value of the unknown,  $d$  the search direction and  $t$  the step. Let  $f$  be the function to minimize. The line search stops when  $t$  verifies

$$f(x + t \cdot d) - f(x) \leq -\alpha t f'(x + t \cdot d)$$

and

$$f(x + \frac{t}{\beta} \cdot d) - f(x) > -\frac{\alpha}{\beta} t f'(x + t \cdot d)$$

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

### Public Member Functions

- [ArmijoLineSearch](#) (Real eps=1e-8, Real alpha=0.05, Real beta=0.65)  
*Default constructor.*
- Real [operator\(\)](#) (const [Problem](#) &P, Real t\_ini)  
*Perform line search.*



## 7.28 Array Class Reference

```
#include <ql/Math/array.hpp>
```

### 7.28.1 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.

#### Tests

construction of arrays is checked in a number of cases

### Constructors, destructor, and assignment

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

### Public Types

- typedef Real \* **iterator**
- typedef const Real \* **const\_iterator**
- typedef boost::reverse\_iterator< iterator > **reverse\_iterator**
- typedef boost::reverse\_iterator< const\_iterator > **const\_reverse\_iterator**

### Public Member Functions

#### 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.*

- const [Array](#) & **operator+=** (const [Array](#) &)
- const [Array](#) & **operator\*=** (Real)

- const [Array](#) & **operator=** (const [Array](#) &)
- const [Array](#) & **operator=** (Real)
- const [Array](#) & **operator \*=** (const [Array](#) &)
- const [Array](#) & **operator \*=** (Real)
- const [Array](#) & **operator/=** (const [Array](#) &)
- const [Array](#) & **operator/=** (Real)

### Element access

- Real [operator\[\]](#) (Size) const  
*read-only*
- Real **at** (Size) const
- Real **front** () const
- Real **back** () const
- Real & [operator\[\]](#) (Size)  
*read-write*
- Real & **at** (Size)
- Real & **front** ()
- Real & **back** ()

### Inspectors

- Size [size](#) () const  
*dimension of the array*
- bool [empty](#) () const  
*whether the array is empty*

### 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.)

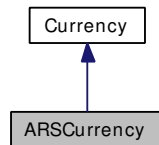
- Real **DotProduct** (const [Array](#) &, const [Array](#) &)
- const [Disposable](#)< [Array](#) > **operator+** (const [Array](#) &v)
- const [Disposable](#)< [Array](#) > **operator-** (const [Array](#) &v)
- const [Disposable](#)< [Array](#) > **operator+** (const [Array](#) &, const [Array](#) &)

- `const Disposable< Array > operator+` (`const Array &`, `Real`)
- `const Disposable< Array > operator+` (`Real`, `const Array &`)
- `const Disposable< Array > operator-` (`const Array &`, `const Array &`)
- `const Disposable< Array > operator-` (`const Array &`, `Real`)
- `const Disposable< Array > operator-` (`Real`, `const Array &`)
- `const Disposable< Array > operator *` (`const Array &`, `const Array &`)
- `const Disposable< Array > operator *` (`const Array &`, `Real`)
- `const Disposable< Array > operator *` (`Real`, `const Array &`)
- `const Disposable< Array > operator/` (`const Array &`, `const Array &`)
- `const Disposable< Array > operator/` (`const Array &`, `Real`)
- `const Disposable< Array > operator/` (`Real`, `const Array &`)
- `const Disposable< Array > Abs` (`const Array &`)
- `const Disposable< Array > Sqrt` (`const Array &`)
- `const Disposable< Array > Log` (`const Array &`)
- `const Disposable< Array > Exp` (`const Array &`)
- `void swap` (`Array &`, `Array &`)
- `std::ostream & operator<<` (`std::ostream &`, `const Array &`)

## 7.29 ARSCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for ARSCurrency:



### 7.29.1 Detailed Description

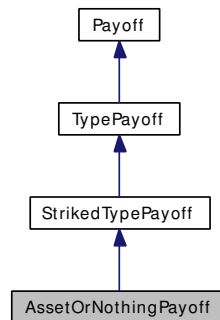
Argentinian peso.

The ISO three-letter code is ARS; the numeric code is 32. It is divided in 100 centavos.

## 7.30 AssetOrNothingPayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for AssetOrNothingPayoff:



### 7.30.1 Detailed Description

Binary asset-or-nothing payoff.

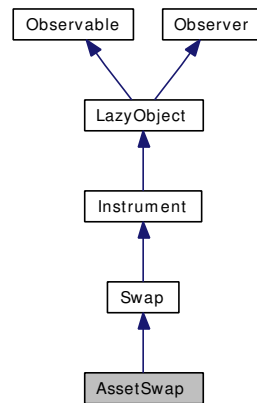
#### Public Member Functions

- **AssetOrNothingPayoff** (Option::Type type, Real strike)
- Real **operator()** (Real price) const
- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.31 AssetSwap Class Reference

```
#include <ql/Instruments/assetswap.hpp>
```

Inheritance diagram for AssetSwap:



### 7.31.1 Detailed Description

Asset swap.

#### Public Member Functions

- **AssetSwap** (bool payFixedRate, const boost::shared\_ptr< [Bond](#) > &bond, Real bondCleanPrice, const [Schedule](#) &floatSchedule, const boost::shared\_ptr< [Xibor](#) > &index, Spread spread, const [DayCounter](#) &floatingDayCount, const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- Spread **fairSpread** () const
- Real **floatingLegBPS** () const
- Real **fairPrice** () const
- Spread **spread** () const
- Real **nominal** () const
- bool **payFixedRate** () const
- const std::vector< boost::shared\_ptr< [CashFlow](#) > > & **bondLeg** () const
- const std::vector< boost::shared\_ptr< [CashFlow](#) > > & **floatingLeg** () const
- void **setupArguments** ([Arguments](#) \*args) const
- void **fetchResults** (const [Results](#) \*) const

#### Classes

- class [arguments](#)  
*Arguments for asset swap calculation*
- class [results](#)  
*Results from simple swap calculation*

## 7.31.2 Member Function Documentation

### 7.31.2.1 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](#).

### 7.31.2.2 void fetchResults (const [Results](#) \*) const [virtual]

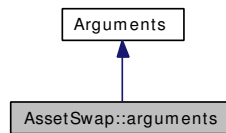
When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

## 7.32 AssetSwap::arguments Class Reference

```
#include <ql/Instruments/assetswap.hpp>
```

Inheritance diagram for AssetSwap::arguments:



### 7.32.1 Detailed Description

Arguments for asset swap calculation

#### Public Member Functions

- void **validate** () const

#### Public Attributes

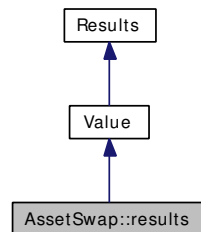
- bool **payFixed**
- Real **nominal**
- std::vector< Time > **fixedResetTimes**
- std::vector< Time > **fixedPayTimes**
- std::vector< Real > **fixedCoupons**
- std::vector< Time > **floatingAccrualTimes**
- std::vector< Time > **floatingResetTimes**
- std::vector< Time > **floatingFixingTimes**
- std::vector< Time > **floatingPayTimes**
- std::vector< Spread > **floatingSpreads**
- Real **currentFloatingCoupon**



## 7.33 AssetSwap::results Class Reference

```
#include <ql/Instruments/assetswap.hpp>
```

Inheritance diagram for AssetSwap::results:



### 7.33.1 Detailed Description

Results from simple swap calculation

#### Public Member Functions

- void **reset** ()

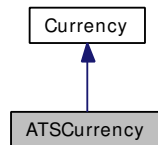
#### Public Attributes

- Real **floatingLegBPS**
- Spread **fairSpread**
- Real **fairPrice**

## 7.34 ATSCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ATSCurrency:



### 7.34.1 Detailed Description

Austrian shilling.

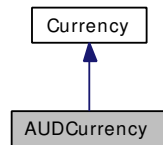
The ISO three-letter code was ATS; the numeric code was 40. It was divided in 100 groschen.

Obsoleted by the Euro since 1999.

## 7.35 AUDCurrency Class Reference

```
#include <ql/Currencies/oceania.hpp>
```

Inheritance diagram for AUDCurrency:



### 7.35.1 Detailed Description

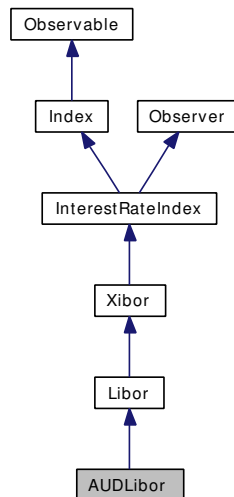
Australian dollar.

The ISO three-letter code is AUD; the numeric code is 36. It is divided into 100 cents.

## 7.36 AUDLibor Class Reference

```
#include <ql/Indexes/audlibor.hpp>
```

Inheritance diagram for AUDLibor:



### 7.36.1 Detailed Description

AUD LIBOR rate

Australian Dollar LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

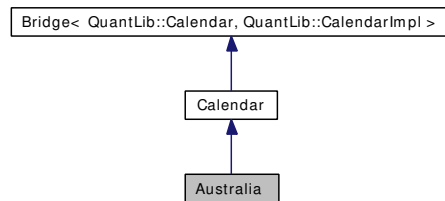
### Public Member Functions

- **AUDLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference, [Integer](#) settlementDays=2)

## 7.37 Australia Class Reference

```
#include <ql/Calendars/australia.hpp>
```

Inheritance diagram for Australia:



### 7.37.1 Detailed Description

Australian calendar.

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)

## 7.38 Average Struct Reference

```
#include <ql/Instruments/asianoption.hpp>
```

### 7.38.1 Detailed Description

placeholder for enumerated averaging types

#### Public Types

- enum Type { Arithmetic, Geometric }

## 7.39 BackwardFlat Class Reference

```
#include <ql/Math/backwardflatinterpolation.hpp>
```

### 7.39.1 Detailed Description

Backward-flat interpolation factory and traits.

#### Public Types

- enum { **global** = 0 }

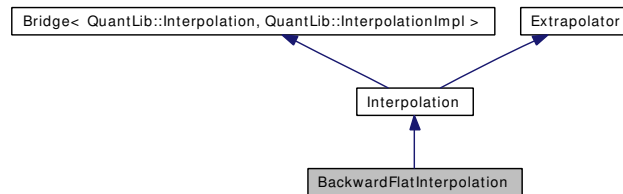
#### Public Member Functions

- template<class I1, class I2> [Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

## 7.40 BackwardFlatInterpolation Class Reference

```
#include <ql/Math/backwardflatinterpolation.hpp>
```

Inheritance diagram for BackwardFlatInterpolation:



### 7.40.1 Detailed Description

Backward-flat interpolation between discrete points.

#### Public Member Functions

- `template<class I1, class I2> BackwardFlatInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)`

### 7.40.2 Constructor & Destructor Documentation

#### 7.40.2.1 [BackwardFlatInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

**Precondition:**

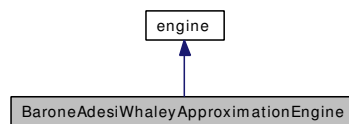
the  $x$  values must be sorted.



## 7.41 BaroneAdesiWhaleyApproximationEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp>
```

Inheritance diagram for BaroneAdesiWhaleyApproximationEngine:



### 7.41.1 Detailed Description

Pricing engine for American options with Barone-Adesi and Whaley approximation (1987)

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

#### Examples:

[EquityOption.cpp](#).

### Public Member Functions

- void `calculate ()` const

### Static Public Member Functions

- static `Real criticalPrice` (const boost::shared\_ptr< `StrikedTypePayoff` > &payoff, `DiscountFactor` riskFreeDiscount, `DiscountFactor` dividendDiscount, `Real` variance, `Real` tolerance=1e-6)

## 7.42 Barrier Struct Reference

```
#include <ql/Instruments/barrieroption.hpp>
```

### 7.42.1 Detailed Description

Placeholder for enumerated barrier types.

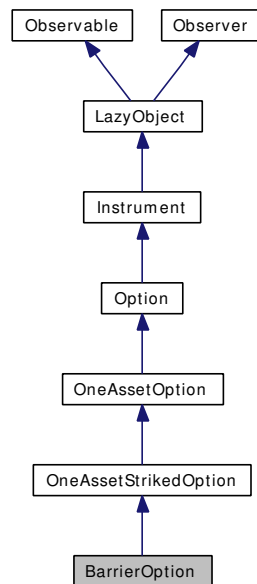
#### Public Types

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

## 7.43 BarrierOption Class Reference

```
#include <ql/Instruments/barrieroption.hpp>
```

Inheritance diagram for BarrierOption:



### 7.43.1 Detailed Description

Barrier option on a single asset.

The analytic pricing engine will be used if none is passed.

**Examples:**

[Replication.cpp](#).

### Public Member Functions

- **BarrierOption** (Barrier::Type barrierType, Real barrier, Real rebate, const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) \*) const

### Protected Attributes

- Barrier::Type **barrierType\_**
- Real **barrier\_**
- Real **rebate\_**

## Classes

- class [arguments](#)  
*Arguments for barrier option calculation*
- class [engine](#)  
*Barrier engine base class*

## 7.43.2 Member Function Documentation

### 7.43.2.1 `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 [OneAssetStrikedOption](#).

## 7.44 BarrierOption::arguments Class Reference

```
#include <ql/Instruments/barrieroption.hpp>
```

### 7.44.1 Detailed Description

Arguments for barrier option calculation

#### Public Member Functions

- void **validate** () const

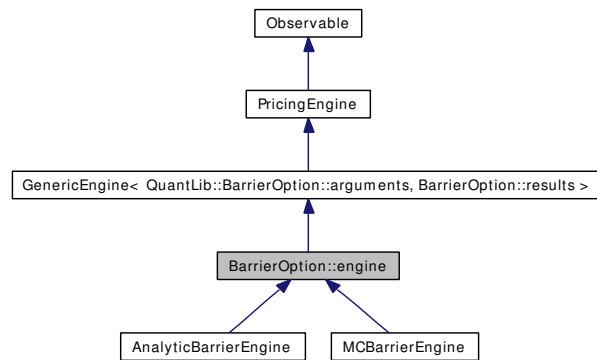
#### Public Attributes

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

## 7.45 BarrierOption::engine Class Reference

```
#include <ql/Instruments/barrieroption.hpp>
```

Inheritance diagram for BarrierOption::engine:



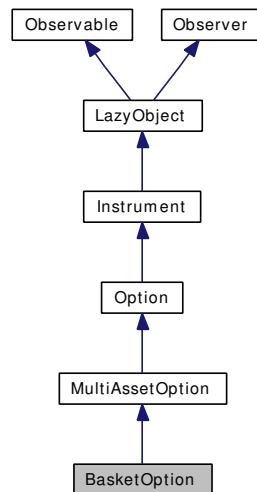
### 7.45.1 Detailed Description

Barrier engine base class

## 7.46 BasketOption Class Reference

```
#include <ql/Instruments/basketoption.hpp>
```

Inheritance diagram for BasketOption:



### 7.46.1 Detailed Description

Basket option on a number of assets.

#### Public Types

- enum **BasketType** { **Min**, **Max** }

#### Public Member Functions

- **BasketOption** (const BasketType basketType, const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [PlainVanillaPayoff](#) > &, const boost::shared\_ptr< [Exercise](#) > &, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) \*) const

#### Classes

- class [arguments](#)  
*Arguments for basket option calculation*
- class [engine](#)  
*Basket option engine base class*

## 7.46.2 Member Function Documentation

### 7.46.2.1 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 [MultiAssetOption](#).



## 7.47 BasketOption::arguments Class Reference

```
#include <ql/Instruments/basketoption.hpp>
```

### 7.47.1 Detailed Description

Arguments for basket option calculation

#### Public Member Functions

- void **validate** () const

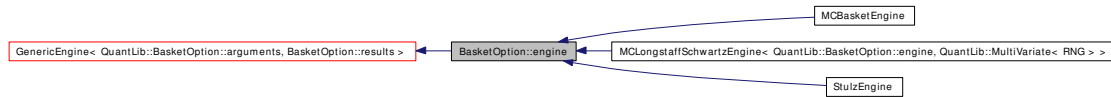
#### Public Attributes

- BasketType **basketType**

## 7.48 BasketOption::engine Class Reference

```
#include <ql/Instruments/basketoption.hpp>
```

Inheritance diagram for BasketOption::engine:



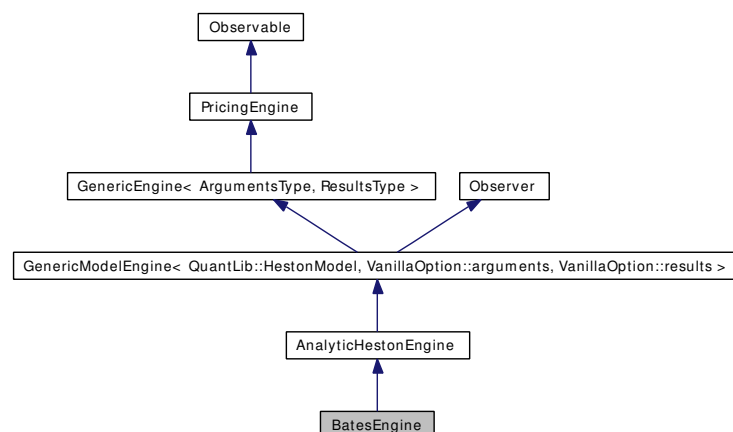
### 7.48.1 Detailed Description

Basket option engine base class

## 7.49 BatesEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/batesengine.hpp>
```

Inheritance diagram for BatesEngine:



### 7.49.1 Detailed Description

Bates model engines based on Fourier transform.

this classes price european options under the following processes

#### 1. Jump-Diffusion with Stochastic Volatility

$$\begin{aligned}
 dS(t, S) &= (r - d - \lambda m)Sdt + \sqrt{v}SdW_1 + (e^J - 1)SdN \\
 dv(t, S) &= \kappa(\theta - v)dt + \sigma\sqrt{v}dW_2 \\
 dW_1dW_2 &= \rho dt
 \end{aligned}$$

$N$  is a Poisson process with the intensity  $\lambda$ . When a jump occurs the magnitude  $J$  has the probability distribution function  $\omega(J)$ .

##### 1.1 Log-Normal Jump Diffusion: [BatesEngine](#)

Logarithm of the jump size  $J$  is normally distributed

$$\omega(J) = \frac{1}{\sqrt{2\pi}\delta^2} \exp\left[-\frac{(J - \nu)^2}{2\delta^2}\right]$$

##### 1.2 Double-Exponential Jump Diffusion: [BatesDoubleExpEngine](#)

The jump size has an asymmetric double exponential distribution

$$\begin{aligned}
 \omega(J) &= p \frac{1}{\eta_u} e^{-\frac{1}{\eta_u} J} 1_{J>0} + q \frac{1}{\eta_d} e^{\frac{1}{\eta_d} J} 1_{J<0} \\
 p + q &= 1
 \end{aligned}$$

#### 2. Stochastic Volatility with Jump Diffusion and Deterministic Jump Intensity

$$\begin{aligned}
 dS(t, S) &= (r - d - \lambda m)Sdt + \sqrt{v}SdW_1 + (e^J - 1)SdN \\
 dv(t, S) &= \kappa(\theta - v)dt + \sigma\sqrt{v}dW_2 \\
 d\lambda(t) &= \kappa_\lambda(\theta_\lambda - \lambda)dt \\
 dW_1dW_2 &= \rho dt
 \end{aligned}$$

2.1 Log-Normal Jump Diffusion with Deterministic Jump Intensity `BatesDetJumpEngine`

2.2 Double-Exponential Jump Diffusion with Deterministic Jump Intensity `BatesDoubleExpDetJumpEngine`

References:

D. Bates, "Jumps and stochastic volatility: exchange rate processes implicit in Deutsche mark options", *Review of Financial Studies* 9, 69-107.

A. Sepp, "Pricing European-Style Options under Jump Diffusion Processes with Stochastic Volatility: Applications of Fourier Transform" (<http://math.ut.ee/~spartak/papers/stochjumpvols.pdf>)

### Tests

the correctness of the returned value is tested by reproducing results available in web/literature, testing against QuantLib's jump diffusion engine and comparison with Black pricing.

### Public Member Functions

- `BatesEngine` (`const boost::shared_ptr< BatesModel > &model`, `Size integrationOrder=64`)

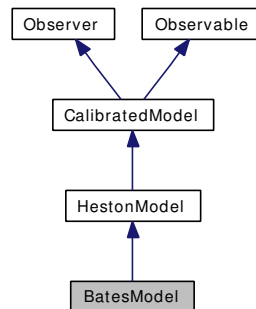
### Protected Member Functions

- `std::complex< Real > jumpDiffusionTerm` (`Real phi`, `Time t`, `Size j`) `const`

## 7.50 BatesModel Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/batesmodel.hpp>
```

Inheritance diagram for BatesModel:



### 7.50.1 Detailed Description

extended versions of Heston model for the stochastic volatility of an asset including jumps.

References: A. Sepp, Pricing European-Style Options under Jump Diffusion Processes with Stochastic Volatility: Applications of Fourier Transform (<http://math.ut.ee/~spartak/papers/stochjumpvols.pdf>)

#### Tests

calibration is tested against known values.

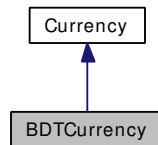
### Public Member Functions

- **BatesModel** (const boost::shared\_ptr< [HestonProcess](#) > &process, [Real](#) lambda=0.1, [Real](#) nu=0.0, [Real](#) delta=0.1)
- [Real](#) **nu** () const
- [Real](#) **delta** () const
- [Real](#) **lambda** () const

## 7.51 BDTCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for BDTCurrency:



### 7.51.1 Detailed Description

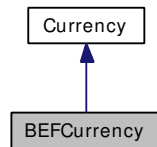
Bangladesh taka.

The ISO three-letter code is BDT; the numeric code is 50. It is divided in 100 paisa.

## 7.52 BEFCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for BEFCurrency:



### 7.52.1 Detailed Description

Belgian franc.

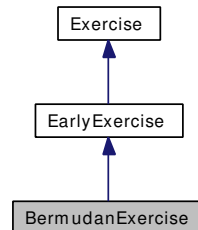
The ISO three-letter code was BEF; the numeric code was 56. It had no subdivisions.

Obsoleted by the Euro since 1999.

## 7.53 BermudanExercise Class Reference

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

Inheritance diagram for BermudanExercise:



### 7.53.1 Detailed Description

Bermudan exercise.

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

#### Todo

it would be nice to have a way for making a Bermudan with one exercise date equivalent to an European

#### Examples:

[BermudanSwaption.cpp](#), and [EquityOption.cpp](#).

### Public Member Functions

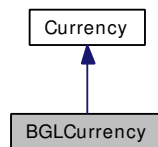
- **BermudanExercise** (const std::vector< [Date](#) > &dates, bool payoffAtExpiry=false)



## 7.54 BGLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for BGLCurrency:



### 7.54.1 Detailed Description

Bulgarian lev.

The ISO three-letter code is BGL; the numeric code is 100. It is divided in 100 stotinki.

## 7.55 Bicubic Class Reference

```
#include <ql/Math/bicubicsplineinterpolation.hpp>
```

### 7.55.1 Detailed Description

bicubic-spline interpolation factory

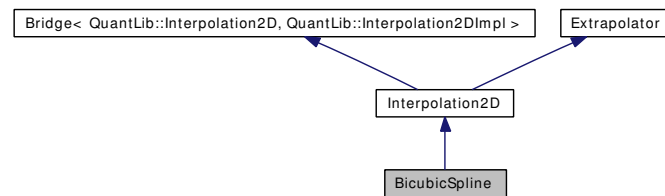
#### Public Member Functions

- `template<class I1, class I2, class M> Interpolation2D interpolate (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &z) const`

## 7.56 BicubicSpline Class Reference

```
#include <ql/Math/bicubicsplineinterpolation.hpp>
```

Inheritance diagram for BicubicSpline:



### 7.56.1 Detailed Description

bicubic-spline interpolation between discrete points

#### Todo

revise end conditions

### Public Member Functions

- `template<class I1, class I2, class M> BicubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)`

### 7.56.2 Constructor & Destructor Documentation

**7.56.2.1** `BicubicSpline (const I1 & xBegin, const I1 & xEnd, const I2 & yBegin, const I2 & yEnd, const M & zData)`

#### Precondition:

the *x* and *y* values must be sorted.

## 7.57 Bilinear Class Reference

```
#include <ql/Math/bilinearinterpolation.hpp>
```

### 7.57.1 Detailed Description

bilinear interpolation factory

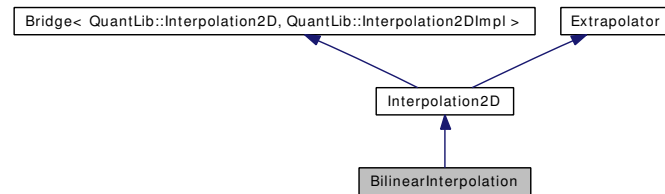
#### Public Member Functions

- `template<class I1, class I2, class M> Interpolation2D interpolate (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &z) const`

## 7.58 BilinearInterpolation Class Reference

```
#include <ql/Math/bilinearinterpolation.hpp>
```

Inheritance diagram for BilinearInterpolation:



### 7.58.1 Detailed Description

bilinear interpolation between discrete points

#### Public Member Functions

- `template<class I1, class I2, class M> BilinearInterpolation(const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)`

### 7.58.2 Constructor & Destructor Documentation

- 7.58.2.1 [BilinearInterpolation](#)(const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, const I2 & *yEnd*, const M & *zData*)

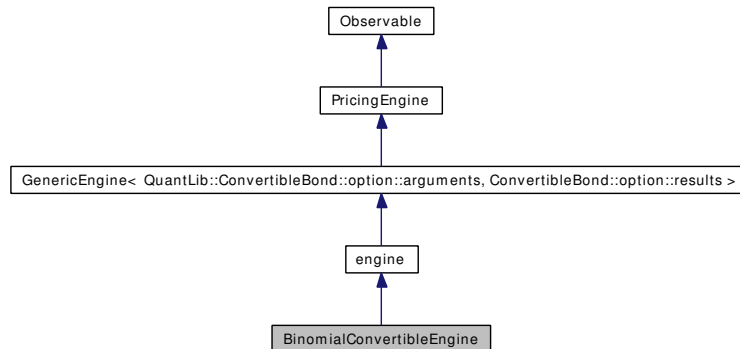
**Precondition:**

the *x* and *y* values must be sorted.

## 7.59 BinomialConvertibleEngine Class Template Reference

```
#include <ql/PricingEngines/Hybrid/binomialconvertibleengine.hpp>
```

Inheritance diagram for BinomialConvertibleEngine:



### 7.59.1 Detailed Description

```
template<class T> class QuantLib::BinomialConvertibleEngine< T >
```

Binomial Tsiveriotis-Fernandes engine for convertible bonds.

Examples:

[ConvertibleBonds.cpp](#).

### Public Member Functions

- **BinomialConvertibleEngine** (Size timeSteps)
- void **calculate** () const

## 7.60 BinomialDistribution Class Reference

```
#include <ql/Math/binomialdistribution.hpp>
```

### 7.60.1 Detailed Description

Binomial probability distribution function.

formula here ... Given an integer k it returns its probability in a Binomial distribution with parameters p and n.

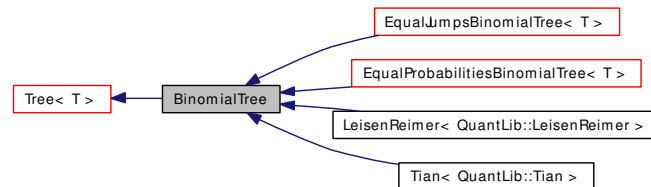
### Public Member Functions

- **BinomialDistribution** (Real p, BigNatural n)
- Real **operator()** (BigNatural k) const

## 7.61 BinomialTree Class Template Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for BinomialTree:



### 7.61.1 Detailed Description

```
template<class T> class QuantLib::BinomialTree< T >
```

Binomial tree base class.

#### Public Types

- enum { **branches** = 2 }

#### Public Member Functions

- **BinomialTree** (const boost::shared\_ptr< [StochasticProcess1D](#) > &process, Time end, Size steps)
- Size **size** (Size i) const
- Size **descendant** (Size, Size index, Size branch) const

#### Protected Attributes

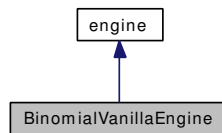
- Real **x0\_**
- Real **driftPerStep\_**
- Time **dt\_**



## 7.62 BinomialVanillaEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/binomialengine.hpp>
```

Inheritance diagram for BinomialVanillaEngine:



### 7.62.1 Detailed Description

```
template<class T> class QuantLib::BinomialVanillaEngine< T >
```

Pricing engine for vanilla options using binomial trees.

#### Tests

the correctness of the returned value is tested by checking it against analytic results.

#### Examples:

[EquityOption.cpp](#).

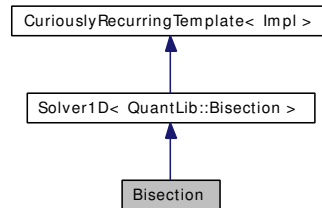
### Public Member Functions

- **BinomialVanillaEngine** (Size timeSteps)
- void **calculate** () const

## 7.63 Bisection Class Reference

```
#include <ql/Solvers1D/bisection.hpp>
```

Inheritance diagram for Bisection:



### 7.63.1 Detailed Description

Bisection 1-D solver

#### Tests

the correctness of the returned values is tested by checking them against known good results.

### Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

## 7.64 BivariateCumulativeNormalDistributionDr78 Class Reference

```
#include <ql/Math/bivariatenormaldistribution.hpp>
```

### 7.64.1 Detailed Description

Cumulative bivariate normal distribution function.

Drezner (1978) algorithm, six decimal places accuracy.

For this implementation see "Option pricing formulas", E.G. Haug, McGraw-Hill 1998

#### Todo

check accuracy of this algorithm and compare with: 1) Drezner, Z., (1978), Computation of the bivariate normal integral, Mathematics of Computation 32, pp. 277-279. 2) Drezner, Z. and Wesolowsky, G. O. (1990) 'On the Computation of the Bivariate Normal Integral', Journal of Statistical Computation and Simulation 35, pp. 101-107. 3) Drezner, Z (1992) Computation of the Multivariate Normal Integral, ACM Transactions on Mathematics Software 18, pp. 450-460. 4) Drezner, Z (1994) Computation of the Trivariate Normal Integral, Mathematics of Computation 62, pp. 289-294. 5) Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.

#### Tests

the correctness of the returned value is tested by checking it against known good results.

### Public Member Functions

- **BivariateCumulativeNormalDistributionDr78** (Real rho)
- Real **operator()** (Real a, Real b) const

## 7.65 BivariateCumulativeNormalDistributionWe04DP Class Reference

```
#include <ql/Math/bivariatenormaldistribution.hpp>
```

### 7.65.1 Detailed Description

Cumulative bivariate normal distribution function (West 2004).

The implementation derives from the article "Better Approximations To Cumulative Normal Distributions", Graeme West, Dec 2004 available at [www.finmod.co.za](http://www.finmod.co.za). Also available in Wilmott Magazine, 2005, (May), 70-76, The main code is a port of the C++ code at [www.finmod.co.za/cumfunctions.zip](http://www.finmod.co.za/cumfunctions.zip).

The algorithm is based on the near double-precision algorithm described in "Numerical Computation of Rectangular Bivariate an Trivariate Normal and t Probabilities", Genz (2004), Statistics and Computing 14, 151-160. (available at [www.sci.wsu.edu/math/faculty/henz/homepage](http://www.sci.wsu.edu/math/faculty/henz/homepage))

The QuantLib implementation mainly differs from the original code in two regards;

- The implementation of the cumulative normal distribution is [QuantLib::CumulativeNormalDistribution](#)
- The arrays XX and W are zero-based

#### Tests

the correctness of the returned value is tested by checking it against known good results.

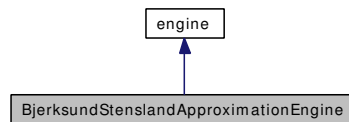
### Public Member Functions

- **BivariateCumulativeNormalDistributionWe04DP** (Real rho)
- Real **operator()** (Real a, Real b) const

## 7.66 BjerksundStenslandApproximationEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/bjerksundstenslandengine.hpp>
```

Inheritance diagram for BjerksundStenslandApproximationEngine:



### 7.66.1 Detailed Description

Pricing engine for American options with Bjerksund and Stensland approximation (1993)

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

#### Examples:

[EquityOption.cpp](#).

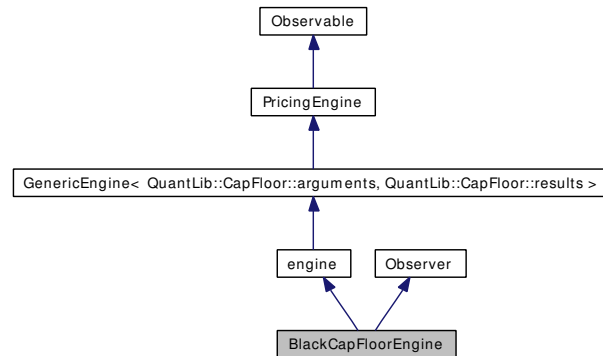
### Public Member Functions

- void **calculate** () const

## 7.67 BlackCapFloorEngine Class Reference

#include <ql/PricingEngines/CapFloor/blackcapfloorengine.hpp>

Inheritance diagram for BlackCapFloorEngine:



### 7.67.1 Detailed Description

Black-formula cap/floor engine.

### Public Member Functions

- **BlackCapFloorEngine** (const [Handle](#)< [Quote](#) > &volatility)
- **BlackCapFloorEngine** (const [Handle](#)< [CapletVolatilityStructure](#) > &)
- void **calculate** () const
- void **update** ()

### 7.67.2 Member Function Documentation

#### 7.67.2.1 void update () [virtual]

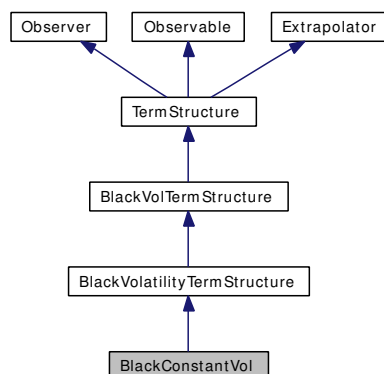
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.68 BlackConstantVol Class Reference

```
#include <ql/Volatilities/blackconstantvol.hpp>
```

Inheritance diagram for BlackConstantVol:



### 7.68.1 Detailed Description

Constant Black volatility, no time-strike dependence.

This class implements the [BlackVolatilityTermStructure](#) interface for a constant Black volatility (no time/strike dependence).

**Examples:**

[ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), and [Replication.cpp](#).

### Public Member Functions

- **BlackConstantVol** (const [Date](#) &referenceDate, Volatility volatility, const [DayCounter](#) &dayCounter)
- **BlackConstantVol** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- **BlackConstantVol** (Integer settlementDays, const [Calendar](#) &, Volatility volatility, const [DayCounter](#) &dayCounter)
- **BlackConstantVol** (Integer settlementDays, const [Calendar](#) &, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)

#### BlackVolTermStructure interface

- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- Time **maxTime** () const  
*the latest time for which the curve can return values*

- Real [minStrike](#) () const  
*the minimum strike for which the term structure can return vols*
- Real [maxStrike](#) () const  
*the maximum strike for which the term structure can return vols*

### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

### Protected Member Functions

- virtual Volatility [blackVolImpl](#) (Time t, Real) const  
*Black volatility calculation.*



## 7.69 BlackFormula Class Reference

```
#include <ql/PricingEngines/blackformula.hpp>
```

### 7.69.1 Detailed Description

Black-formula calculator.

#### Bug

When the variance is null, division by zero occur during the calculation of delta, delta forward, gamma, gamma forward, rho, dividend rho, vega, and strike sensitivity.

#### Examples:

[DiscreteHedging.cpp](#).

### Public Member Functions

- **BlackFormula** (Real forward, [DiscountFactor](#) discount, Real variance, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff)
- Real **value** () const
- Real **delta** (Real spot) const
- Real [elasticity](#) (Real spot) const

*Sensitivity in percent to a percent movement in the underlying.*

- Real **gamma** (Real spot) const
- Real **deltaForward** () const
- Real [elasticityForward](#) () const

*Sensitivity in percent to a percent movement in the forward price.*

- Real **gammaForward** () const
- Real **theta** (Real spot, [Time](#) maturity) const
- Real **thetaPerDay** (Real spot, [Time](#) maturity) const
- Real **vega** ([Time](#) maturity) const
- Real **rho** ([Time](#) maturity) const
- Real **dividendRho** ([Time](#) maturity) const
- Real [itmCashProbability](#) () const
- Real [itmAssetProbability](#) () const
- Real **strikeSensitivity** () const
- Real **alpha** () const
- Real **beta** () const

### Friends

- class **Calculator**

## 7.69.2 Member Function Documentation

### 7.69.2.1 Real itmCashProbability () const

Probability of being in the money in the bond martingale measure. It is a risk-neutral probability, not the real world probability.

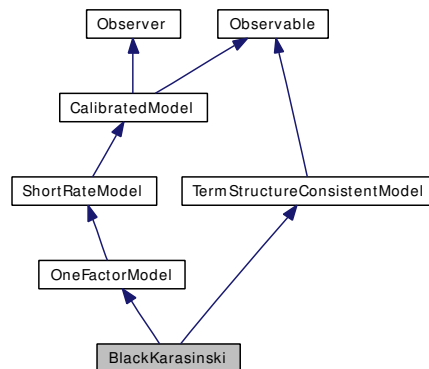
### 7.69.2.2 Real itmAssetProbability () const

Probability of being in the money in the asset martingale measure. It is a risk-neutral probability, not the real world probability.

## 7.70 BlackKarasinski Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp>
```

Inheritance diagram for BlackKarasinski:



### 7.70.1 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.

**Examples:**

[BermudanSwaption.cpp](#).

### Public Member Functions

- **BlackKarasinski** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a=0.1, [Real](#) sigma=0.1)
- `boost::shared_ptr< ShortRateDynamics > dynamics () const`  
*returns the short-rate dynamics*
- `boost::shared_ptr< NumericalMethod > tree (const TimeGrid &grid) const`  
*Return by default a trinomial recombining tree.*

### Classes

- class [Dynamics](#)  
*Short-rate dynamics in the Black-Karasinski model.*

## 7.71 BlackKarasinski::Dynamics Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp>
```

### 7.71.1 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.

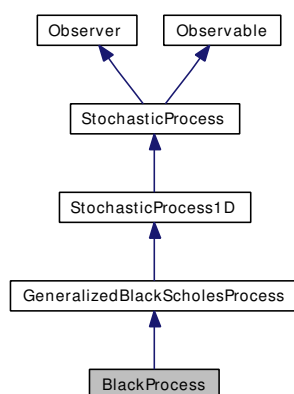
### Public Member Functions

- **Dynamics** (const [Parameter](#) &fitting, [Real](#) alpha, [Real](#) sigma)
- **Real variable** ([Time](#) t, [Rate](#) r) const
- **Real shortRate** ([Time](#) t, [Real](#) x) const

## 7.72 BlackProcess Class Reference

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Inheritance diagram for BlackProcess:



### 7.72.1 Detailed Description

Black (1976) stochastic process.

This class describes the stochastic process for a forward or futures contract given by

$$dS(t, S) = \frac{\sigma(t, S)^2}{2} dt + \sigma dW_t.$$

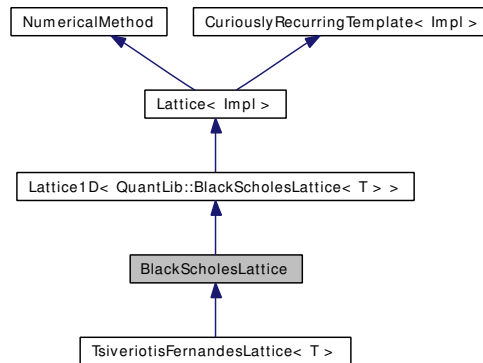
### Public Member Functions

- **BlackProcess** (const [Handle< Quote >](#) &x0, const [Handle< YieldTermStructure >](#) &risk-FreeTS, const [Handle< BlackVolTermStructure >](#) &blackVolTS, const boost::shared\_ptr< discretization > &d=boost::shared\_ptr< discretization >(new [EulerDiscretization](#)))

## 7.73 BlackScholesLattice Class Template Reference

```
#include <ql/Lattices/bsmlattice.hpp>
```

Inheritance diagram for BlackScholesLattice:



### 7.73.1 Detailed Description

```
template<class T> class QuantLib::BlackScholesLattice< T >
```

Simple binomial lattice approximating the Black-Scholes model.

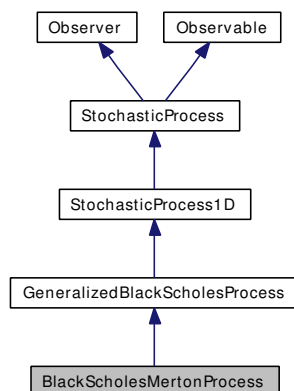
#### Public Member Functions

- **BlackScholesLattice** (const boost::shared\_ptr< T > &tree, Rate riskFreeRate, Time end, Size steps)
- Size **size** (Size i) const
- DiscountFactor **discount** (Size, Size) const
- void **stepback** (Size i, const Array &values, Array &newValues) const
- Real **underlying** (Size i, Size index) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

## 7.74 BlackScholesMertonProcess Class Reference

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Inheritance diagram for BlackScholesMertonProcess:



### 7.74.1 Detailed Description

Merton (1973) extension to the Black-Scholes stochastic process.

This class describes the stochastic process for a stock or stock index paying a continuous dividend yield given by

$$dS(t, S) = (r(t) - q(t) - \frac{\sigma(t, S)^2}{2})dt + \sigma dW_t.$$

**Examples:**

[ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), and [EquityOption.cpp](#).

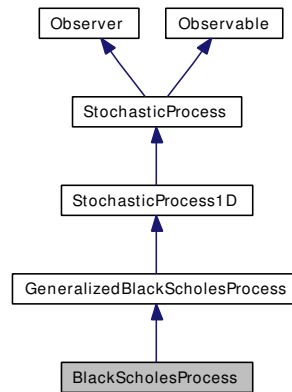
### Public Member Functions

- **BlackScholesMertonProcess** (const [Handle< Quote >](#) &x0, const [Handle< YieldTermStructure >](#) &dividendTS, const [Handle< YieldTermStructure >](#) &riskFreeTS, const [Handle< BlackVolTermStructure >](#) &blackVolTS, const boost::shared\_ptr< discretization > &d=boost::shared\_ptr< discretization >(new [EulerDiscretization](#)))

## 7.75 BlackScholesProcess Class Reference

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Inheritance diagram for BlackScholesProcess:



### 7.75.1 Detailed Description

Black-Scholes (1973) stochastic process.

This class describes the stochastic process for a stock given by

$$dS(t, S) = \left(r(t) - \frac{\sigma(t, S)^2}{2}\right)dt + \sigma dW_t.$$

Examples:

[Replication.cpp](#).

### Public Member Functions

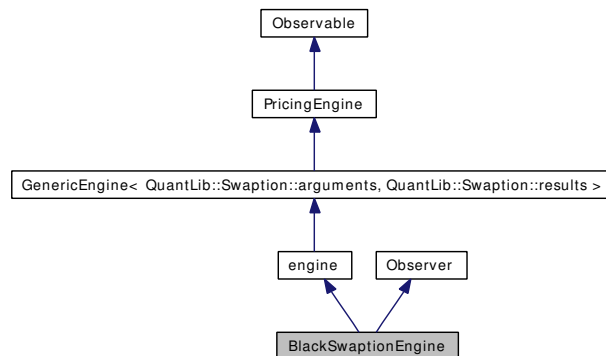
- **BlackScholesProcess** (const [Handle< Quote >](#) &x0, const [Handle< YieldTermStructure >](#) &riskFreeTS, const [Handle< BlackVolTermStructure >](#) &blackVolTS, const boost::shared\_ptr< discretization > &d=boost::shared\_ptr< discretization >(new [EulerDiscretization](#)))



## 7.76 BlackSwaptionEngine Class Reference

```
#include <ql/PricingEngines/Swaption/blackswaptionengine.hpp>
```

Inheritance diagram for BlackSwaptionEngine:



### 7.76.1 Detailed Description

Black-formula swaption engine.

#### Warning

The engine assumes that the exercise date equals the start date of the passed swap.

### Public Member Functions

- **BlackSwaptionEngine** (const [Handle](#)< [Quote](#) > &volatility)
- **BlackSwaptionEngine** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &)
- void **calculate** () const
- void **update** ()

### 7.76.2 Member Function Documentation

#### 7.76.2.1 void update () [virtual]

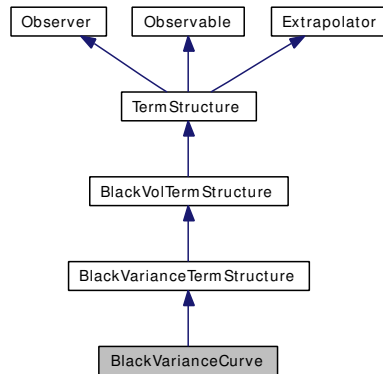
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.77 BlackVarianceCurve Class Reference

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Inheritance diagram for BlackVarianceCurve:



### 7.77.1 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](#).

#### Todo

check time extrapolation

### Public Member Functions

- **BlackVarianceCurve** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &dates, const std::vector< [Volatility](#) > &blackVolCurve, const [DayCounter](#) &dayCounter, bool forceMonotoneVariance=true)

#### BlackVolTermStructure interface

- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- Real **minStrike** () const  
*the minimum strike for which the term structure can return vols*
- Real **maxStrike** () const

*the maximum strike for which the term structure can return vols*

### Modifiers

- `template<class Interpolator> void setInterpolation (const Interpolator &i=Interpolator())`

### Visitability

- `virtual void accept (AcyclicVisitor &)`

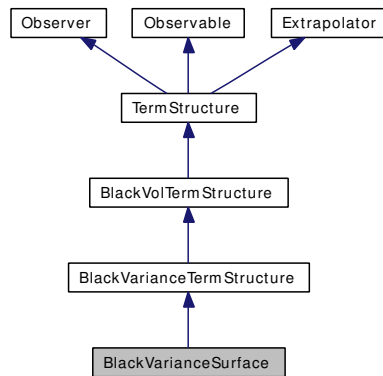
### Protected Member Functions

- `virtual Real blackVarianceImpl (Time t, Real) const`  
*Black variance calculation.*

## 7.78 BlackVarianceSurface Class Reference

```
#include <ql/Volatilities/blackvariancesurface.hpp>
```

Inheritance diagram for BlackVarianceSurface:



### 7.78.1 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.

#### Todo

check time extrapolation

### Public Types

- enum `Extrapolation` { `ConstantExtrapolation`, `InterpolatorDefaultExtrapolation` }

### Public Member Functions

- **BlackVarianceSurface** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &dates, const std::vector< [Real](#) > &strikes, const [Matrix](#) &blackVolMatrix, const [DayCounter](#) &dayCounter, Extrapolation lowerExtrapolation=InterpolatorDefaultExtrapolation, Extrapolation upperExtrapolation=InterpolatorDefaultExtrapolation)

#### BlackVolTermStructure interface

- [DayCounter](#) `dayCounter` () const  
*the day counter used for date/time conversion*
- [Date](#) `maxDate` () const  
*the latest date for which the curve can return values*

- Real [minStrike](#) () const  
*the minimum strike for which the term structure can return vols*
- Real [maxStrike](#) () const  
*the maximum strike for which the term structure can return vols*

### Modifiers

- template<class Interpolator> void **setInterpolation** (const Interpolator &i=Interpolator())

### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

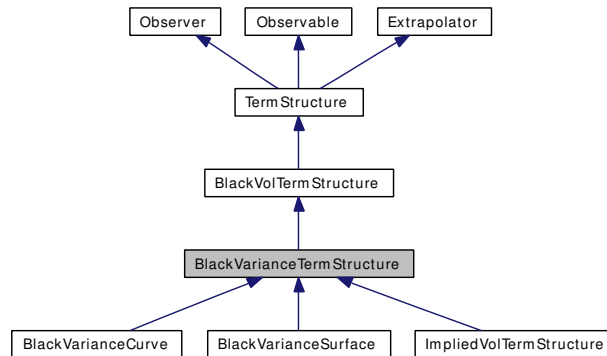
### Protected Member Functions

- virtual Real [blackVarianceImpl](#) (Time t, Real strike) const  
*Black variance calculation.*

## 7.79 BlackVarianceTermStructure Class Reference

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

Inheritance diagram for BlackVarianceTermStructure:



### 7.79.1 Detailed Description

Black variance term structure.

This abstract class acts as an adapter to VolTermStructure allowing the programmer to implement only the `blackVarianceImpl(Time, Real, bool)` method in derived classes.

Volatility are assumed to be expressed on an annual basis.

### Public Member Functions

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

### Protected Member Functions

- Volatility **blackVolImpl** (Time maturity, Real strike) const

## 7.79.2 Constructor & Destructor Documentation

### 7.79.2.1 [BlackVarianceTermStructure](#) ()

default constructor

#### Warning

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

### 7.79.2.2 [BlackVarianceTermStructure](#) ()

default constructor

#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.79.3 Member Function Documentation

### 7.79.3.1 [Volatility](#) `blackVolImpl (Time maturity, Real strike) const` [protected, virtual]

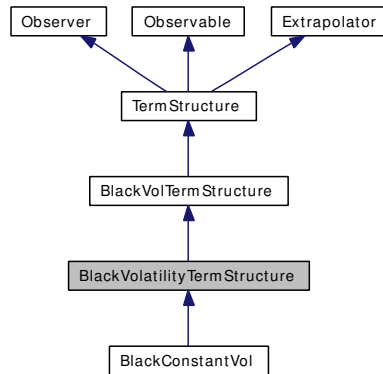
Returns the volatility for the given strike and date calculating it from the variance.

Implements [BlackVolTermStructure](#).

## 7.80 BlackVolatilityTermStructure Class Reference

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

Inheritance diagram for BlackVolatilityTermStructure:



### 7.80.1 Detailed Description

Black-volatility term structure.

This abstract class acts as an adapter to [BlackVolTermStructure](#) allowing the programmer to implement only the `blackVolImpl(Time, Real, bool)` method in derived classes.

Volatility are assumed to be expressed on an annual basis.

### Public Member Functions

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

### Protected Member Functions

- Real **blackVarianceImpl** (Time maturity, Real strike) const

### 7.80.2 Constructor & Destructor Documentation

#### 7.80.2.1 [BlackVolatilityTermStructure](#) ()

default constructor

#### Warning

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.



### 7.80.2.2 [BlackVolatilityTermStructure](#) ()

default constructor

#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.80.3 Member Function Documentation

### 7.80.3.1 Real [blackVarianceImpl](#) (Time *maturity*, Real *strike*) const [protected, virtual]

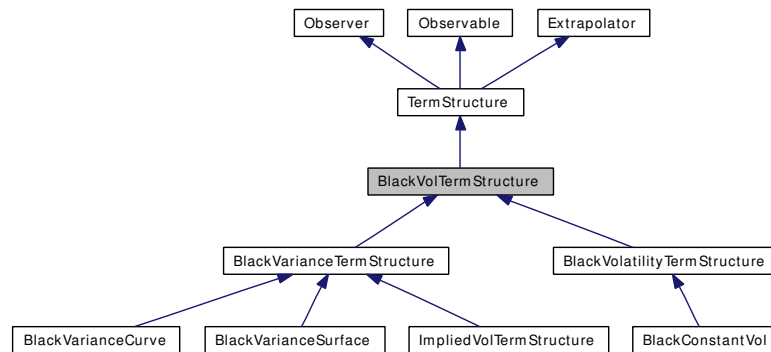
Returns the variance for the given strike and date calculating it from the volatility.

Implements [BlackVolTermStructure](#).

## 7.81 BlackVolTermStructure Class Reference

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

Inheritance diagram for BlackVolTermStructure:



### 7.81.1 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

- **Volatility blackVol** (const [Date](#) &maturity, Real strike, bool extrapolate=false) const  
*present (a.k.a spot) volatility*
- **Volatility blackVol** (Time maturity, Real strike, bool extrapolate=false) const  
*present (a.k.a spot) volatility*
- Real **blackVariance** (const [Date](#) &maturity, Real strike, bool extrapolate=false) const  
*present (a.k.a spot) variance*
- Real **blackVariance** (Time maturity, Real strike, bool extrapolate=false) const  
*present (a.k.a spot) variance*
- **Volatility blackForwardVol** (const [Date](#) &date1, const [Date](#) &date2, Real strike, bool extrapolate=false) const  
*future (a.k.a. forward) volatility*
- **Volatility blackForwardVol** (Time time1, Time time2, Real strike, bool extrapolate=false) const  
*future (a.k.a. forward) volatility*

- Real [blackForwardVariance](#) (const [Date](#) &date1, const [Date](#) &date2, Real strike, bool extrapolate=false) const  
*future (a.k.a. forward) variance*
- Real [blackForwardVariance](#) (Time time1, Time time2, Real strike, bool extrapolate=false) const  
*future (a.k.a. forward) variance*

### Limits

- virtual Real [minStrike](#) () const=0  
*the minimum strike for which the term structure can return vols*
- virtual Real [maxStrike](#) () const=0  
*the maximum strike for which the term structure can return vols*

### Visitability

- virtual void [accept](#) ([AcyclicVisitor](#) &)

## Protected Member Functions

### Calculations

These methods must be implemented in derived classes to perform the actual volatility calculations. When they are called, range check has already been performed; therefore, they must assume that extrapolation is required.

- virtual Real [blackVarianceImpl](#) (Time t, Real strike) const =0  
*Black variance calculation.*
- virtual [Volatility](#) [blackVollImpl](#) (Time t, Real strike) const =0  
*Black volatility calculation.*

## 7.81.2 Constructor & Destructor Documentation

### 7.81.2.1 [BlackVolTermStructure](#) ()

default constructor

### Warning

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

### 7.81.2.2 [BlackVolTermStructure](#) ()

default constructor

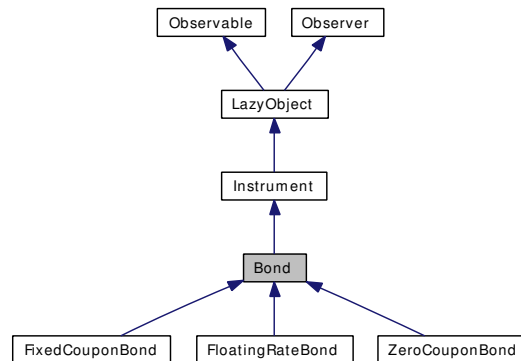
#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.82 Bond Class Reference

```
#include <ql/Instruments/bond.hpp>
```

Inheritance diagram for Bond:



### 7.82.1 Detailed Description

Base bond class.

Derived classes must fill the uninitialized data members.

#### Warning

Most methods assume that the cashflows are stored sorted by date, the redemption being the last one.

#### Tests

- price/yield calculations are cross-checked for consistency.
- price/yield calculations are checked against known good values.

## Public Member Functions

### Inspectors

- [Date](#) **settlementDate** () const
- [Date](#) **maturityDate** () const
- [Date](#) **firstCouponDate** () const
- const std::vector< boost::shared\_ptr< [CashFlow](#) > > & **cashflows** () const
- const boost::shared\_ptr< [CashFlow](#) > & **redemption** () const
- const [Calendar](#) & **calendar** () const
- [BusinessDayConvention](#) **accrualConvention** () const
- [BusinessDayConvention](#) **paymentConvention** () const
- Real **faceAmount** () const
- const [DayCounter](#) & **dayCounter** () const
- [Frequency](#) **frequency** () const
- boost::shared\_ptr< [YieldTermStructure](#) > **discountCurve** () const

### Calculations

- Real `cleanPrice` () const  
*theoretical clean price*
- Real `dirtyPrice` () const  
*theoretical dirty price*
- `Rate yield` (Compounding compounding, Real accuracy=1.0e-8, Size max-Evaluations=100) const  
*theoretical bond yield*
- Real `cleanPrice` (`Rate` yield, Compounding compounding, `Date` settlementDate=`Date`()) const  
*clean price given a yield and settlement date*
- Real `dirtyPrice` (`Rate` yield, Compounding compounding, `Date` settlementDate=`Date`()) const  
*dirty price given a yield and settlement date*
- `Rate yield` (Real cleanPrice, Compounding compounding, `Date` settlementDate=`Date`()), Real accuracy=1.0e-8, Size maxEvaluations=100) const  
*yield given a (clean) price and settlement date*
- Real `accruedAmount` (`Date` d=`Date`()) const  
*accrued amount at a given date*
- bool `isExpired` () const  
*returns whether the instrument is still tradable.*

## Protected Member Functions

- `Bond` (Real faceAmount, const `DayCounter` &dayCount, const `Calendar` &calendar, `BusinessDayConvention` accrualConvention, `BusinessDayConvention` paymentConvention, Integer settlementDays, const `Handle`< `YieldTermStructure` > &discountCurve=`Handle`< `YieldTermStructure` >())
- `Bond` (const `DayCounter` &dayCount, const `Calendar` &calendar, `BusinessDayConvention` accrualConvention, `BusinessDayConvention` paymentConvention, Integer settlementDays, const `Handle`< `YieldTermStructure` > &discountCurve=`Handle`< `YieldTermStructure` >())
- void `performCalculations` () const

## Protected Attributes

- Integer `settlementDays_`
- `Calendar` `calendar_`
- `BusinessDayConvention` `accrualConvention_`
- `BusinessDayConvention` `paymentConvention_`
- Real `faceAmount_`
- `DayCounter` `dayCount_`
- `Date` `issueDate_`
- `Date` `datedDate_`
- `Date` `maturityDate_`
- `Frequency` `frequency_`
- std::vector< boost::shared\_ptr< `CashFlow` > > `cashflows_`
- `Handle`< `YieldTermStructure` > `discountCurve_`

## 7.82.2 Constructor & Destructor Documentation

7.82.2.1 **Bond** (const **DayCounter** & *dayCount*, const **Calendar** & *calendar*, **BusinessDayConvention** *accrualConvention*, **BusinessDayConvention** *paymentConvention*, Integer *settlementDays*, const **Handle**< **YieldTermStructure** > & *discountCurve* = **Handle**< **YieldTermStructure** >()) [protected]

### Deprecated

use constructor with face amount instead

## 7.82.3 Member Function Documentation

7.82.3.1 **const std::vector< boost::shared\_ptr< CashFlow > > & cashflows () const**

### Warning

the returned vector includes the redemption as the last cash flow.

7.82.3.2 **Real cleanPrice () const**

theoretical clean price

The default bond settlement is used for calculation.

### Warning

the theoretical price calculated from a flat term structure might differ slightly from the price calculated from the corresponding yield by means of the other overload of this function. If the price from a constant yield is desired, it is advisable to use such other overload.

7.82.3.3 **Real dirtyPrice () const**

theoretical dirty price

The default bond settlement is used for calculation.

### Warning

the theoretical price calculated from a flat term structure might differ slightly from the price calculated from the corresponding yield by means of the other overload of this function. If the price from a constant yield is desired, it is advisable to use such other overload.

7.82.3.4 **Rate** *yield (Compounding *compounding*, Real *accuracy* = 1.0e-8, Size *maxEvaluations* = 100) const*

theoretical bond yield

The default bond settlement and theoretical price are used for calculation.

#### 7.82.3.5 Real cleanPrice (**Rate** *yield*, *Compounding compounding*, **Date** *settlementDate* = **Date()**) const

clean price given a yield and settlement date

The default bond settlement is used if no date is given.

#### 7.82.3.6 Real dirtyPrice (**Rate** *yield*, *Compounding compounding*, **Date** *settlementDate* = **Date()**) const

dirty price given a yield and settlement date

The default bond settlement is used if no date is given.

#### 7.82.3.7 **Rate** *yield* (**Real** *cleanPrice*, *Compounding compounding*, **Date** *settlementDate* = **Date()**, **Real** *accuracy* = 1.0e-8, **Size** *maxEvaluations* = 100) const

yield given a (clean) price and settlement date

The default bond settlement is used if no date is given.

#### 7.82.3.8 Real accruedAmount (**Date** *d* = **Date()**) const

accrued amount at a given date

The default bond settlement is used if no date is given.

#### 7.82.3.9 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.

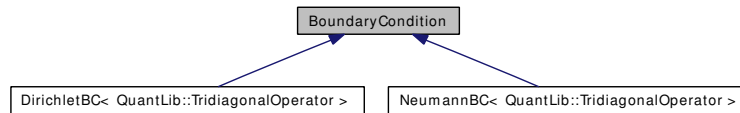
Reimplemented from [Instrument](#).



## 7.83 BoundaryCondition Class Template Reference

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Inheritance diagram for BoundaryCondition:



### 7.83.1 Detailed Description

```
template<class Operator> class QuantLib::BoundaryCondition< Operator >
```

Abstract boundary condition class for finite difference problems.

#### Public Types

- enum [Side](#) { None, Upper, Lower }
- typedef Operator **operator\_type**
- typedef Operator::array\_type **array\_type**

#### Public Member Functions

- virtual void [applyBeforeApplying](#) (operator\_type &) const=0
- virtual void [applyAfterApplying](#) (array\_type &) const =0
- virtual void [applyBeforeSolving](#) (operator\_type &, array\_type &rhs) const=0
- virtual void [applyAfterSolving](#) (array\_type &) const =0
- virtual void [setTime](#) (Time t)=0

### 7.83.2 Member Enumeration Documentation

#### 7.83.2.1 enum [Side](#)

##### [Todo](#)

Generalize for n-dimensional conditions

### 7.83.3 Member Function Documentation

#### 7.83.3.1 virtual void [applyBeforeApplying](#) (operator\_type &) 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.

#### 7.83.3.2 virtual void [applyAfterApplying](#) (array\_type &) const [pure virtual]

This method modifies an array  $u$  so that it satisfies the given condition.

**7.83.3.3** `virtual void applyBeforeSolving (operator_type &, array_type & 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.

**7.83.3.4** `virtual void applyAfterSolving (array_type &) const` [pure virtual]

This method modifies an array  $u$  so that it satisfies the given condition.

**7.83.3.5** `virtual void setTime (Time t)` [pure virtual]

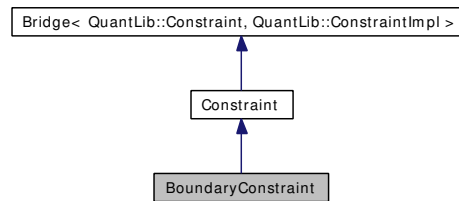
This method sets the current time for time-dependent boundary conditions.

Implemented in [NeumannBC](#), and [DirichletBC](#).

## 7.84 BoundaryConstraint Class Reference

```
#include <ql/Optimization/constraint.hpp>
```

Inheritance diagram for BoundaryConstraint:



### 7.84.1 Detailed Description

Constraint imposing all arguments to be in [low,high]

#### Public Member Functions

- `BoundaryConstraint` ([Real](#) low, [Real](#) high)

## 7.85 BoxMullerGaussianRng Class Template Reference

```
#include <ql/RandomNumbers/boxmullergaussianrng.hpp>
```

### 7.85.1 Detailed Description

```
template<class RNG> class QuantLib::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 [Sample](#)< Real > **sample\_type**
- typedef RNG **urng\_type**

### Public Member Functions

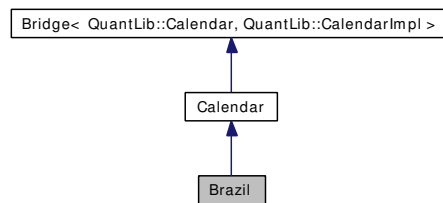
- **BoxMullerGaussianRng** (const RNG &uniformGenerator)
- [sample\\_type next](#) () const

*returns a sample from a Gaussian distribution*

## 7.86 Brazil Class Reference

```
#include <ql/Calendars/brazil.hpp>
```

Inheritance diagram for Brazil:



### 7.86.1 Detailed Description

Brazilian calendar.

Banking holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Tiradentes's Day, April 21th
- Labour Day, May 1st
- Independence Day, September 21th
- Nossa Sra. Aparecida Day, October 12th
- Dead Day, October 2nd
- Republic Day, November 15th
- Christmas, December 25th
- Passion of Christ
- Carnival
- Corpus Christi

#### Tests

the correctness of the returned results is tested against a list of known holidays.

### Public Types

- enum [Market](#) { [Settlement](#) }
- Brazilian calendars.*

## Public Member Functions

- **Brazil** ([Market](#) market=[Settlement](#))

## 7.86.2 Member Enumeration Documentation

### 7.86.2.1 enum [Market](#)

Brazilian calendars.

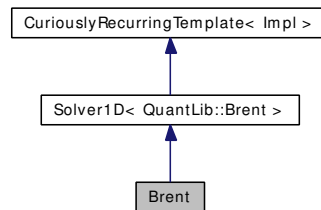
#### Enumerator:

*Settlement* generic settlement calendar

## 7.87 Brent Class Reference

```
#include <ql/Solvers1D/brent.hpp>
```

Inheritance diagram for Brent:



### 7.87.1 Detailed Description

Brent 1-D solver

#### Tests

the correctness of the returned values is tested by checking them against known good results.

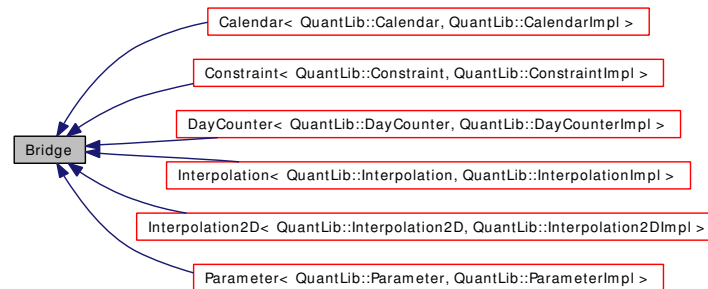
### Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

## 7.88 Bridge Class Template Reference

```
#include <ql/Patterns/bridge.hpp>
```

Inheritance diagram for Bridge:



### 7.88.1 Detailed Description

```
template<class T, class T_impl> class QuantLib::Bridge< T, T_impl >
```

The Bridge 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 **empty** () const

### Protected Member Functions

- **Bridge** (const boost::shared\_ptr< Impl > &impl=boost::shared\_ptr< Impl >())

### Protected Attributes

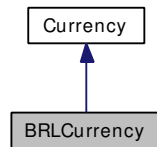
- boost::shared\_ptr< Impl > **impl\_**



## 7.89 BRLCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for BRLCurrency:



### 7.89.1 Detailed Description

Brazilian real.

The ISO three-letter code is BRL; the numeric code is 986. It is divided in 100 centavos.

## 7.90 BrownianBridge Class Template Reference

```
#include <ql/MonteCarlo/brownianbridge.hpp>
```

### 7.90.1 Detailed Description

```
template<class GSG> class QuantLib::BrownianBridge< GSG >
```

Builds Wiener process paths using Gaussian variates.

For more details: "Monte Carlo Methods in Finance" by P. Jäckel, section 10.8.3

#### Note:

this class does not work if the diffusion term of the underlying stochastic process is asset-dependent.

### Public Types

- typedef [Sample](#)< std::vector< Real > > **sample\_type**

### Public Member Functions

- [BrownianBridge](#) (const GSG &generator)  
*normalised (unit time, unit variance) Wiener process paths*
- [BrownianBridge](#) (Time length, Size timeSteps, const GSG &generator)  
*unit variance Wiener process paths*
- [BrownianBridge](#) (const [TimeGrid](#) &timeGrid, const GSG &generator)  
*unit variance Wiener process paths*
- [BrownianBridge](#) (const std::vector< Real > &sigma, const [TimeGrid](#) &timeGrid, const GSG &generator)  
*general Wiener process paths*
- [BrownianBridge](#) (const boost::shared\_ptr< [BlackVolTermStructure](#) > &, const [TimeGrid](#) &timeGrid, const GSG &generator)
- [BrownianBridge](#) (const boost::shared\_ptr< [StochasticProcess1D](#) > &, const [TimeGrid](#) &timeGrid, const GSG &generator)

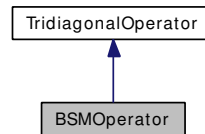
#### inspectors

- const sample\_type & **next** () const
- const sample\_type & **last** () const
- Size **size** () const
- const [TimeGrid](#) & **timeGrid** () const

## 7.91 BSMOperator Class Reference

```
#include <ql/FiniteDifferences/bsmoperator.hpp>
```

Inheritance diagram for BSMOperator:



### 7.91.1 Detailed Description

Black-Scholes-Merton differential operator.

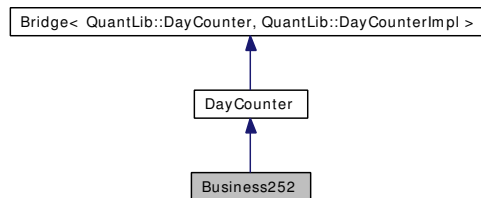
#### Public Member Functions

- **BSMOperator** (Size size, [Real](#) dx, [Rate](#) r, [Rate](#) q, [Volatility](#) sigma)
- **BSMOperator** (const [Array](#) &grid, const boost::shared\_ptr< [GeneralizedBlackScholes-Process](#) > &, [Time](#) residualTime)

## 7.92 Business252 Class Reference

```
#include <ql/DayCounters/business252.hpp>
```

Inheritance diagram for Business252:



### 7.92.1 Detailed Description

Business/252 day count convention.

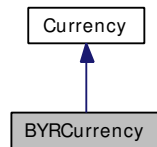
#### Public Member Functions

- **Business252** ([Calendar](#) c)

## 7.93 BYRCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for BYRCurrency:



### 7.93.1 Detailed Description

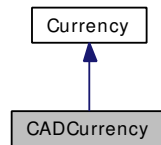
Belarussian ruble.

The ISO three-letter code is BYR; the numeric code is 974. It has no subdivisions.

## 7.94 CADCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for CADCurrency:



### 7.94.1 Detailed Description

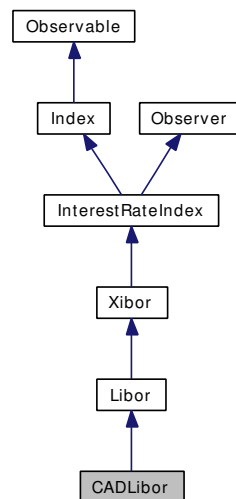
Canadian dollar.

The ISO three-letter code is CAD; the numeric code is 124. It is divided into 100 cents.

## 7.95 CADLibor Class Reference

```
#include <ql/Indexes/cadlibor.hpp>
```

Inheritance diagram for CADLibor:



### 7.95.1 Detailed Description

CAD LIBOR rate

Canadian Dollar LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

#### Warning

This is the rate fixed in London by BBA. Use CDOR if you're interested in the Canadian fixing by IDA.

### Public Member Functions

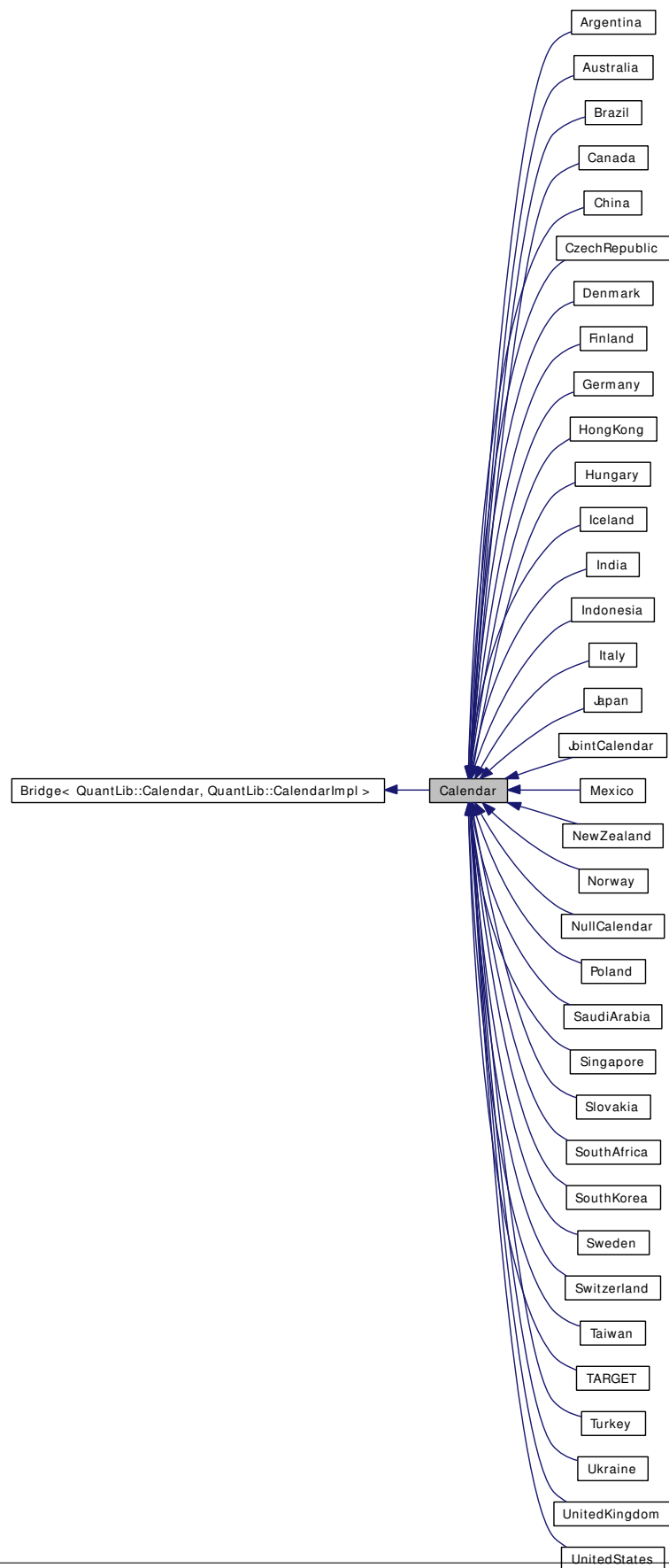
- **CADLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference, [Integer](#) settlementDays=2)

## 7.96 Calendar Class Reference

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

Inheritance diagram for Calendar:





## 7.96.1 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.

A calendar should be defined for specific exchange holiday schedule or for general country holiday schedule. Legacy city holiday schedule calendars will be moved to the exchange/country convention.

### Tests

the methods for adding and removing holidays are tested by inspecting the calendar before and after their invocation.

### Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [FRA.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

## Calendar interface

- `std::string name () const`  
*Returns the name of the calendar.*
- `bool isBusinessDay (const Date &d) const`
- `bool isHoliday (const Date &d) const`
- `bool isWeekend (Weekday w) const`
- `bool isEndOfMonth (const Date &d) const`
- `Date endOfMonth (const Date &d) const`  
*last business day of the month to which the given date belongs*
- `void addHoliday (const Date &)`
- `void removeHoliday (const Date &)`
- `Date adjust (const Date &, BusinessDayConvention convention=Following, const Date &origin=Date()) const`
- `Date advance (const Date &, Integer n, TimeUnit unit, BusinessDayConvention convention=Following, bool endOfMonth=false) const`
- `Date advance (const Date &date, const Period &period, BusinessDayConvention convention=Following, bool endOfMonth=false) const`
- `BigInteger businessDaysBetween (const Date &from, const Date &to, bool includeFirst=true, bool includeLast=false) const`
- `static std::vector< Date > holidayList (const Calendar &calendar, const Date &from, const Date &to, bool includeWeekEnds=false)`  
*Returns the holidays between two dates.*

## Public Member Functions

- `Calendar ()`

## Protected Member Functions

- [Calendar](#) (const boost::shared\_ptr< [CalendarImpl](#) > &impl)

## Related Functions

(Note that these are not member functions.)

- bool [operator==](#) (const [Calendar](#) &, const [Calendar](#) &)
- bool [operator!=](#) (const [Calendar](#) &, const [Calendar](#) &)
- std::ostream & [operator<<](#) (std::ostream &, const [Calendar](#) &)

## Classes

- class [OrthodoxImpl](#)  
*partial calendar implementation*
- class [WesternImpl](#)  
*partial calendar implementation*

## 7.96.2 Constructor & Destructor Documentation

### 7.96.2.1 [Calendar](#) ()

This default constructor returns a calendar with a null implementation, which is therefore unusable except as a placeholder.

### 7.96.2.2 [Calendar](#) (const boost::shared\_ptr< [CalendarImpl](#) > & impl) [protected]

This protected constructor will only be invoked by derived classes which define a given [Calendar](#) implementation

## 7.96.3 Member Function Documentation

### 7.96.3.1 [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.

### 7.96.3.2 [bool isBusinessDay \(const \[Date\]\(#\) & d\) const](#)

Returns true iff the date is a business day for the given market.

### 7.96.3.3 `bool isHoliday (const Date & d) const`

Returns true iff the date is a holiday for the given market.

### 7.96.3.4 `bool isWeekend (Weekday w) const`

Returns true iff the weekday is part of the weekend for the given market.

### 7.96.3.5 `bool isEndOfMonth (const Date & d) const`

Returns true iff the date is last business day for the month in given market.

### 7.96.3.6 `void addHoliday (const Date &)`

Adds a date to the set of holidays for the given calendar.

### 7.96.3.7 `void removeHoliday (const Date &)`

Removes a date from the set of holidays for the given calendar.

### 7.96.3.8 `Date adjust (const Date &, BusinessDayConvention convention = Following, const Date & origin = Date()) const`

Adjusts a non-business day to the appropriate near business day with respect to the given convention.

**Examples:**

[ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

### 7.96.3.9 `Date advance (const Date &, Integer n, TimeUnit unit, BusinessDayConvention convention = Following, bool endOfMonth = false) const`

Advances the given date of the given number of business days and returns the result.

**Note:**

The input date is not modified.

**Examples:**

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [FRA.cpp](#), and [swapvaluation.cpp](#).

### 7.96.3.10 `Date advance (const Date & date, const Period & period, BusinessDayConvention convention = Following, bool endOfMonth = false) const`

Advances the given date as specified by the given period and returns the result.

**Note:**

The input date is not modified.

**7.96.3.11** `BigInteger businessDaysBetween (const Date & from, const Date & to, bool includeFirst = true, bool includeLast = false) const`

Calculates the number of business days between two given dates and returns the result.

## 7.96.4 Friends And Related Function Documentation

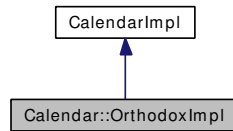
**7.96.4.1** `bool operator== (const Calendar &, const Calendar &) [related]`

Returns true iff the two calendars belong to the same derived class.

## 7.97 Calendar::OrthodoxImpl Class Reference

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

Inheritance diagram for Calendar::OrthodoxImpl:



### 7.97.1 Detailed Description

partial calendar implementation

This class provides the means of determining the Orthodox Easter Monday for a given year, as well as specifying Saturdays and Sundays as weekend days.

#### Protected Member Functions

- bool **isWeekend** ([Weekday](#)) const

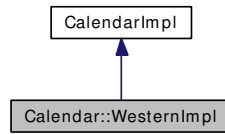
#### Static Protected Member Functions

- static Day **easterMonday** (Year)  
*expressed relative to first day of year*

## 7.98 Calendar::WesternImpl Class Reference

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

Inheritance diagram for Calendar::WesternImpl:



### 7.98.1 Detailed Description

partial calendar implementation

This class provides the means of determining the Easter Monday for a given year, as well as specifying Saturdays and Sundays as weekend days.

#### Protected Member Functions

- `bool isWeekend (Weekday) const`

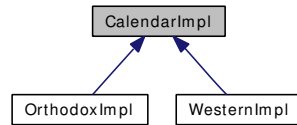
#### Static Protected Member Functions

- `static Day easterMonday (Year)`  
*expressed relative to first day of year*

## 7.99 CalendarImpl Class Reference

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

Inheritance diagram for CalendarImpl:



### 7.99.1 Detailed Description

abstract base class for calendar implementations

#### Public Member Functions

- virtual std::string **name** () const=0
- virtual bool **isBusinessDay** (const [Date](#) &) const =0
- virtual bool **isWeekend** ([Weekday](#)) const=0

#### Public Attributes

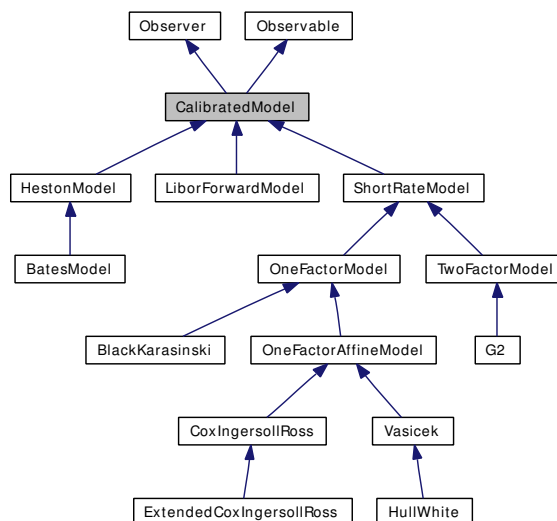
- std::set< [Date](#) > **addedHolidays**
- std::set< [Date](#) > **removedHolidays**



## 7.100 CalibratedModel Class Reference

```
#include <ql/ShortRateModels/model.hpp>
```

Inheritance diagram for CalibratedModel:



### 7.100.1 Detailed Description

Calibrated model class.

#### Public Member Functions

- **CalibratedModel** (Size nArguments)
- void **update** ()
- void **calibrate** (const std::vector< boost::shared\_ptr< CalibrationHelper > > &, OptimizationMethod &method, const Constraint &constraint=Constraint(), const std::vector< Real > &weights=std::vector< Real >())

*Calibrate to a set of market instruments (caps/swaptions).*

- const boost::shared\_ptr< Constraint > & **constraint** () const
- Disposable< Array > **params** () const

*Returns array of arguments on which calibration is done.*

- virtual void **setParams** (const Array &params)

#### Protected Member Functions

- virtual void **generateArguments** ()

## Protected Attributes

- `std::vector< Parameter > arguments_`
- `boost::shared_ptr< Constraint > constraint_`

## Friends

- `class CalibrationFunction`

## 7.100.2 Member Function Documentation

### 7.100.2.1 `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](#).

### 7.100.2.2 `void calibrate(const std::vector< boost::shared_ptr< CalibrationHelper > > &, OptimizationMethod & method, const Constraint & constraint = Constraint(), const std::vector< Real > & weights = std::vector< Real >())`

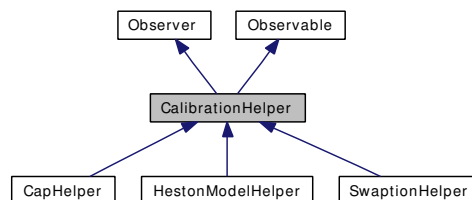
Calibrate to a set of market instruments (caps/swaptions).

An additional constraint can be passed which must be satisfied in addition to the constraints of the model.

## 7.101 CalibrationHelper Class Reference

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

Inheritance diagram for CalibrationHelper:



### 7.101.1 Detailed Description

liquid market instrument used during calibration

#### Public Member Functions

- **CalibrationHelper** (const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [YieldTermStructure](#) > &termStructure, bool calibrateVolatility=false)
- void [update](#) ()
- Real [marketValue](#) () const  
*returns the actual price of the instrument (from volatility)*
- virtual Real [modelValue](#) () const=0  
*returns the price of the instrument according to the model*
- virtual Real [calibrationError](#) ()  
*returns the error resulting from the model valuation*
- virtual void **addTimesTo** (std::list< [Time](#) > &times) const=0
- Volatility [impliedVolatility](#) (Real targetValue, Real accuracy, Size maxEvaluations, Volatility minVol, Volatility maxVol) const  
*Black volatility implied by the model.*
- virtual Real [blackPrice](#) (Volatility volatility) const=0  
*Black price given a volatility.*
- void **setPricingEngine** (const boost::shared\_ptr< [PricingEngine](#) > &engine)

#### Protected Attributes

- Real [marketValue\\_](#)
- [Handle](#)< [Quote](#) > [volatility\\_](#)
- [Handle](#)< [YieldTermStructure](#) > [termStructure\\_](#)
- boost::shared\_ptr< [PricingEngine](#) > [engine\\_](#)

## 7.101.2 Member Function Documentation

### 7.101.2.1 `void update ()` [virtual]

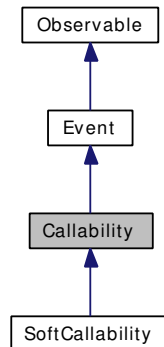
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.102 Callability Class Reference

```
#include <ql/Instruments/callabilityschedule.hpp>
```

Inheritance diagram for Callability:



### 7.102.1 Detailed Description

instrument callability

Examples:

[ConvertibleBonds.cpp](#).

### Public Types

- enum [Type](#) { [Call](#), [Put](#) }  
*type of the callability*

### Public Member Functions

- [Callability](#) (const [Price](#) &price, [Type](#) type, const [Date](#) &date)
- const [Price](#) & [price](#) () const
- [Type](#) [type](#) () const
- [Date](#) [date](#) () const  
*returns the date at which the event occurs*

### Classes

- class [Price](#)  
*amount to be paid upon callability*

## 7.103 Callability::Price Class Reference

```
#include <ql/Instruments/callabilityschedule.hpp>
```

### 7.103.1 Detailed Description

amount to be paid upon callability

Examples:

[ConvertibleBonds.cpp](#).

### Public Types

- enum **Type** { **Dirty**, **Clean** }

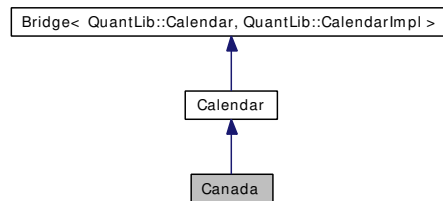
### Public Member Functions

- **Price** (Real amount, Type type)
- Real **amount** () const
- Type **type** () const

## 7.104 Canada Class Reference

```
#include <ql/Calendars/canada.hpp>
```

Inheritance diagram for Canada:



### 7.104.1 Detailed Description

Canadian calendar.

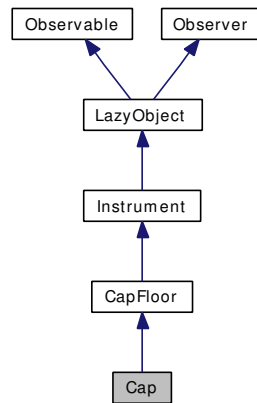
Holidays:

- 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)

## 7.105 Cap Class Reference

```
#include <ql/Instruments/capfloor.hpp>
```

Inheritance diagram for Cap:



### 7.105.1 Detailed Description

Concrete cap class.

#### Public Member Functions

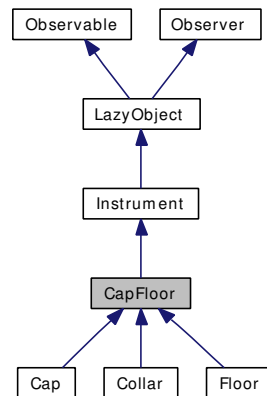
- **Cap** (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &floatingLeg, const std::vector< [Rate](#) > &exerciseRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared\_ptr< [PricingEngine](#) > &engine)



## 7.106 CapFloor Class Reference

```
#include <ql/Instruments/capfloor.hpp>
```

Inheritance diagram for CapFloor:



### 7.106.1 Detailed Description

Base class for cap-like instruments.

#### Tests

- the correctness of the returned value is tested by checking that the price of a cap (resp. floor) decreases (resp. increases) with the strike rate.
- the relationship between the values of caps, floors and the resulting collars is checked.
- the put-call parity between the values of caps, floors and swaps is checked.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the correctness of the returned value is tested by checking it against a known good value.

### Public Types

- enum **Type** { **Cap**, **Floor**, **Collar** }

### Public Member Functions

- **CapFloor** (Type type, const std::vector< boost::shared\_ptr< [CashFlow](#) > > &floatingLeg, const std::vector< Rate > &capRates, const std::vector< Rate > &floorRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared\_ptr< [PricingEngine](#) > &engine)
- void [setupArguments](#) ([Arguments](#) \*) const
- [Volatility](#) [impliedVolatility](#) ([Real](#) price, [Real](#) accuracy=1.0e-4, Size maxEvaluations=100, Volatility minVol=QL\_MIN\_VOLATILITY, Volatility 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< boost::shared\_ptr< [CashFlow](#) > > & `leg ()` const
- const std::vector< Rate > & `capRates ()` const
- const std::vector< Rate > & `floorRates ()` const
- const std::vector< boost::shared\_ptr< [CashFlow](#) > > & `floatingLeg ()` const

### Classes

- class [arguments](#)  
*Arguments for cap/floor calculation*
- class [engine](#)  
*base class for cap/floor engines*
- class [results](#)  
*Results from cap/floor calculation*

## 7.106.2 Member Function Documentation

### 7.106.2.1 void `setupArguments (Arguments *)` const [virtual]

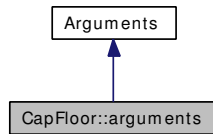
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

## 7.107 CapFloor::arguments Class Reference

```
#include <ql/Instruments/capfloor.hpp>
```

Inheritance diagram for CapFloor::arguments:



### 7.107.1 Detailed Description

Arguments for cap/floor calculation

#### Public Member Functions

- void **validate** () const

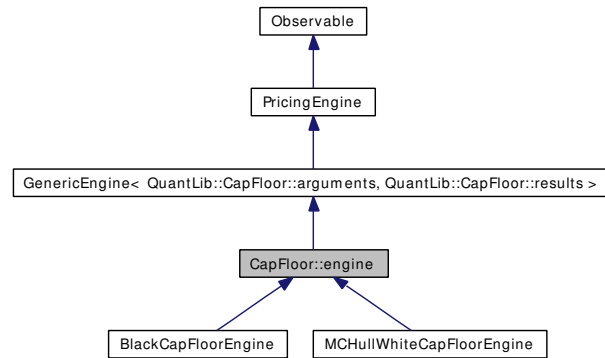
#### Public Attributes

- CapFloor::Type **type**
- std::vector< Time > **startTimes**
- std::vector< [Date](#) > **fixingDates**
- 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< [Real](#) > **gearings**
- std::vector< DiscountFactor > **discounts**
- std::vector< [Real](#) > **nominals**

## 7.108 CapFloor::engine Class Reference

`#include <ql/Instruments/capfloor.hpp>`

Inheritance diagram for CapFloor::engine:



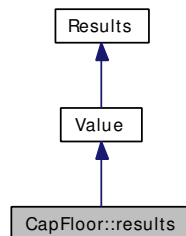
### 7.108.1 Detailed Description

base class for cap/floor engines

## 7.109 CapFloor::results Class Reference

```
#include <ql/Instruments/capfloor.hpp>
```

Inheritance diagram for CapFloor::results:



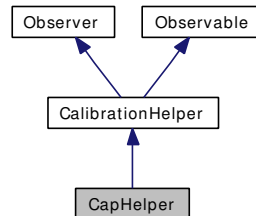
### 7.109.1 Detailed Description

Results from cap/floor calculation

## 7.110 CapHelper Class Reference

```
#include <ql/ShortRateModels/CalibrationHelpers/caphelper.hpp>
```

Inheritance diagram for CapHelper:



### 7.110.1 Detailed Description

calibration helper for ATM cap

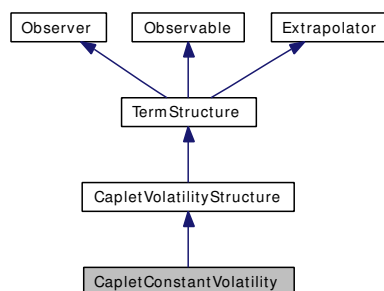
#### Public Member Functions

- **CapHelper** (const [Period](#) &length, const [Handle](#)< [Quote](#) > &volatility, const boost::shared\_ptr< [Xibor](#) > &index, [Frequency](#) fixedLegFrequency, const [DayCounter](#) &fixedLegDayCounter, bool includeFirstSwaplet, const [Handle](#)< [YieldTermStructure](#) > &termStructure, bool calibrateVolatility=false)
- virtual void **addTimesTo** (std::list< Time > &times) const
- virtual [Real](#) **modelValue** () const  
*returns the price of the instrument according to the model*
- virtual [Real](#) **blackPrice** (Volatility volatility) const  
*Black price given a volatility.*

## 7.111 CapletConstantVolatility Class Reference

```
#include <ql/Volatilities/capletconstantvol.hpp>
```

Inheritance diagram for CapletConstantVolatility:



### 7.111.1 Detailed Description

Constant caplet volatility, no time-strike dependence.

#### Public Member Functions

- **CapletConstantVolatility** (const [Date](#) &referenceDate, Volatility volatility, const [DayCounter](#) &dayCounter)
- **CapletConstantVolatility** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- **CapletConstantVolatility** (Volatility volatility, const [DayCounter](#) &dayCounter)
- **CapletConstantVolatility** (const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- [Time](#) **maxTime** () const  
*the latest time for which the curve can return values*
- [Real](#) **minStrike** () const  
*the minimum strike for which the term structure can return vols*
- [Real](#) **maxStrike** () const  
*the maximum strike for which the term structure can return vols*

#### TermStructure interface

- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*

## Protected Member Functions

### CapletVolatilityStructure interface

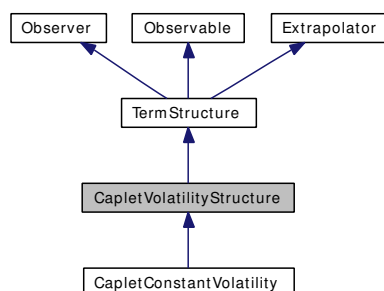
- Volatility [volatilityImpl](#) (Time t, Rate) const  
*implements the actual volatility calculation in derived classes*



## 7.112 CapletVolatilityStructure Class Reference

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

Inheritance diagram for CapletVolatilityStructure:



### 7.112.1 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

#### Volatility and Variance

- **Volatility volatility** (const [Date](#) &start, Rate strike, bool extrapolate=false) const  
*returns the volatility for a given start date and strike rate*
- **Volatility volatility** (Time t, Rate strike, bool extrapolate=false) const  
*returns the volatility for a given start time and strike rate*
- **Real blackVariance** (const [Date](#) &start, Rate strike, bool extrapolate=false) const  
*returns the black variance for a given start date and strike rate*
- **Real blackVariance** (Time t, Rate strike, bool extrapolate=false) const  
*returns the black variance for a given start time and strike rate*

#### Limits

- virtual **Real minStrike** () const=0  
*the minimum strike for which the term structure can return vols*
- virtual **Real maxStrike** () const=0  
*the maximum strike for which the term structure can return vols*

## Protected Member Functions

- virtual [Volatility volatilityImpl](#) (Time length, Rate strike) const=0  
*implements the actual volatility calculation in derived classes*

## 7.112.2 Constructor & Destructor Documentation

### 7.112.2.1 [CapletVolatilityStructure](#) ()

default constructor

#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

### 7.112.2.2 [CapletVolatilityStructure](#) ()

default constructor

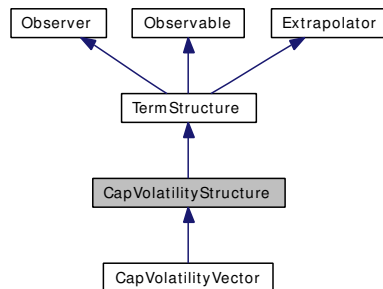
#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.113 CapVolatilityStructure Class Reference

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

Inheritance diagram for CapVolatilityStructure:



### 7.113.1 Detailed Description

Cap/floor term-volatility structure.

This class is purely abstract and defines the interface of concrete structures which will be derived from this one.

### Public Member Functions

#### Volatility

- **Volatility volatility** (const [Date](#) &end, Rate strike, bool extrapolate=false) const
- **Volatility volatility** (const [Period](#) &length, Rate strike, bool extrapolate=false) const  
*returns the volatility for a given cap/floor length and strike rate*
- **Volatility volatility** (Time t, Rate strike, bool extrapolate=false) const  
*returns the volatility for a given end time and strike rate*

#### Limits

- virtual **Real minStrike** () const=0  
*the minimum strike for which the term structure can return vols*
- virtual **Real maxStrike** () const=0  
*the maximum strike for which the term structure can return vols*

### Protected Member Functions

- virtual **Volatility volatilityImpl** (Time length, Rate strike) const=0  
*implements the actual volatility calculation in derived classes*

## 7.113.2 Constructor & Destructor Documentation

### 7.113.2.1 [CapVolatilityStructure](#) ()

default constructor

#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

### 7.113.2.2 [CapVolatilityStructure](#) ()

default constructor

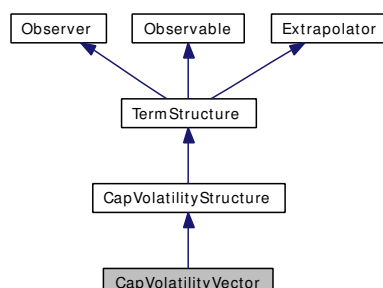
#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.114 CapVolatilityVector Class Reference

```
#include <ql/Volatilities/capflatvolvector.hpp>
```

Inheritance diagram for CapVolatilityVector:



### 7.114.1 Detailed Description

Cap/floor at-the-money term-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

- **CapVolatilityVector** (const [Date](#) &settlementDate, const std::vector< [Period](#) > &lengths, const std::vector< Volatility > &volatilities, const [DayCounter](#) &dayCounter)
- **CapVolatilityVector** (Integer settlementDays, const [Calendar](#) &calendar, const std::vector< [Period](#) > &lengths, const std::vector< Volatility > &volatilities, const [DayCounter](#) &dayCounter)
- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- Time **maxTime** () const  
*the latest time for which the curve can return values*
- [Real](#) **minStrike** () const  
*the minimum strike for which the term structure can return vols*
- [Real](#) **maxStrike** () const  
*the maximum strike for which the term structure can return vols*
- void **update** ()

## 7.114.2 Member Function Documentation

### 7.114.2.1 `void update ()` [virtual]

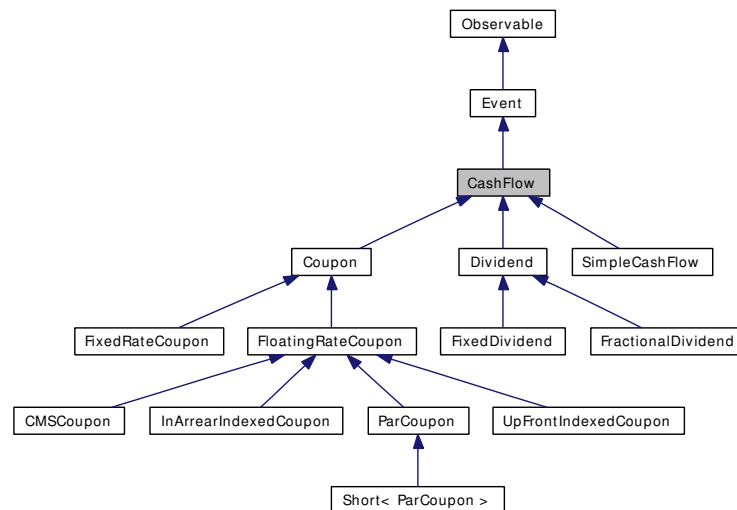
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

## 7.115 CashFlow Class Reference

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

Inheritance diagram for CashFlow:



### 7.115.1 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 **Real amount** () const=0  
*returns the amount of the cash flow*
- virtual **Date date** () const=0  
*Note:*  
*This is inherited from the event class*

#### Visitability

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

### 7.115.2 Member Function Documentation

#### 7.115.2.1 virtual **Real 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 [Dividend](#), [FixedDividend](#), [FractionalDividend](#), [FixedRateCoupon](#), [FloatingRateCoupon](#), and [SimpleCashFlow](#).



## 7.116 Cashflows Class Reference

```
#include <ql/CashFlows/analysis.hpp>
```

### 7.116.1 Detailed Description

cashflows analysis functions

#### Todo

add tests

### Static Public Member Functions

- static [Real](#) [npv](#) (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &, const [Handle](#)< [YieldTermStructure](#) > &)  
*NPV of the cash flows.*
- static [Real](#) [npv](#) (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &, const [InterestRate](#) &, [Date](#) settlementDate=[Date](#)())  
*NPV of the cash flows.*
- static [Real](#) [bps](#) (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &, const [Handle](#)< [YieldTermStructure](#) > &)  
*Basis-point sensitivity of the cash flows.*
- static [Real](#) [bps](#) (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &, const [InterestRate](#) &, [Date](#) settlementDate=[Date](#)())  
*Basis-point sensitivity of the cash flows.*
- static [Rate](#) [irr](#) (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &, [Real](#) marketPrice, const [DayCounter](#) &dayCounter, Compounding compounding, [Frequency](#) frequency=No-Frequency, [Date](#) settlementDate=[Date](#)(), [Real](#) tolerance=1.0e-10, Size maxIterations=10000, [Rate](#) guess=0.05)  
*Internal rate of return.*
- static [Time](#) [duration](#) (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &, const [InterestRate](#) &y, Duration::Type type=Duration::Modified, [Date](#) settlementDate=[Date](#)())  
*Cash-flow duration.*
- static [Real](#) [convexity](#) (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &, const [InterestRate](#) &y, [Date](#) settlementDate=[Date](#)())  
*Cash-flow convexity.*

### 7.116.2 Member Function Documentation

- 7.116.2.1 static [Real](#) [npv](#) (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &, const [Handle](#)< [YieldTermStructure](#) > &) [static]

NPV of the cash flows.

The NPV is the sum of the cash flows, each discounted according to the given term structure.

**7.116.2.2** static **Real** npv (const std::vector< boost::shared\_ptr< **CashFlow** > > &, const **InterestRate** &, **Date** settlementDate = **Date**()) [static]

NPV of the cash flows.

The NPV is the sum of the cash flows, each discounted according to the given constant interest rate. The result is affected by the choice of the interest-rate compounding and the relative frequency and day counter.

**7.116.2.3** static **Real** bps (const std::vector< boost::shared\_ptr< **CashFlow** > > &, const **Handle**< **YieldTermStructure** > &) [static]

Basis-point sensitivity of the cash flows.

The result is the change in NPV due to a uniform 1-basis-point change in the rate paid by the cash flows. The change for each coupon is discounted according to the given term structure.

**7.116.2.4** static **Real** bps (const std::vector< boost::shared\_ptr< **CashFlow** > > &, const **InterestRate** &, **Date** settlementDate = **Date**()) [static]

Basis-point sensitivity of the cash flows.

The result is the change in NPV due to a uniform 1-basis-point change in the rate paid by the cash flows. The change for each coupon is discounted according to the given constant interest rate. The result is affected by the choice of the interest-rate compounding and the relative frequency and day counter.

**7.116.2.5** static **Rate** irr (const std::vector< boost::shared\_ptr< **CashFlow** > > &, **Real** marketPrice, const **DayCounter** & dayCounter, *Compounding compounding*, **Frequency** frequency = NoFrequency, **Date** settlementDate = **Date**(), **Real** tolerance = 1.0e-10, **Size** maxIterations = 10000, **Rate** guess = 0.05) [static]

Internal rate of return.

The IRR is the interest rate at which the NPV of the cash flows equals the given market price. The function verifies the theoretical existence of an IRR and numerically establishes the IRR to the desired precision.

**7.116.2.6** static **Time** duration (const std::vector< boost::shared\_ptr< **CashFlow** > > &, const **InterestRate** & y, **Duration::Type** type = **Duration::Modified**, **Date** settlementDate = **Date**()) [static]

Cash-flow duration.

The simple duration of a string of cash flows is defined as

$$D_{\text{simple}} = \frac{\sum t_i c_i B(t_i)}{\sum c_i B(t_i)}$$

where  $c_i$  is the amount of the  $i$ -th cash flow,  $t_i$  is its payment time, and  $B(t_i)$  is the corresponding discount according to the passed yield.

The modified duration is defined as

$$D_{\text{modified}} = -\frac{1}{P} \frac{\partial P}{\partial y}$$

where  $P$  is the present value of the cash flows according to the given IRR  $y$ .

The Macaulay duration is defined for a compounded IRR as

$$D_{\text{Macaulay}} = \left(1 + \frac{y}{N}\right) D_{\text{modified}}$$

where  $y$  is the IRR and  $N$  is the number of cash flows per year.

**7.116.2.7 static Real convexity** (const std::vector< boost::shared\_ptr< CashFlow > > &, const InterestRate &  $y$ , Date settlementDate = Date()) [static]

Cash-flow convexity.

The convexity of a string of cash flows is defined as

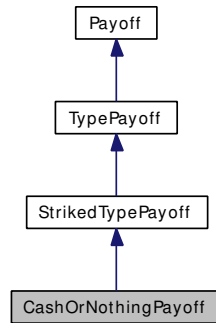
$$C = \frac{1}{P} \frac{\partial^2 P}{\partial y^2}$$

where  $P$  is the present value of the cash flows according to the given IRR  $y$ .

## 7.117 CashOrNothingPayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for CashOrNothingPayoff:



### 7.117.1 Detailed Description

Binary cash-or-nothing payoff.

Examples:

[Replication.cpp](#).

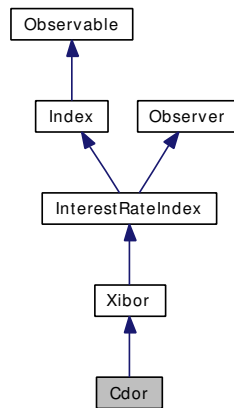
### Public Member Functions

- **CashOrNothingPayoff** (Option::Type type, Real strike, Real cashPayoff)
- Real **operator()** (Real price) const
- Real **cashPayoff** () const
- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.118 Cdor Class Reference

```
#include <ql/Indexes/cdor.hpp>
```

Inheritance diagram for Cdor:



### 7.118.1 Detailed Description

CDOR rate

Canadian Dollar Offered Rate fixed by IDA.

#### Warning

This is the rate fixed in [Canada](#) by IDA. Use [CADLibor](#) if you're interested in the London fixing by BBA.

#### Todo

check settlement days and day-count convention.

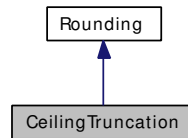
### Public Member Functions

- **Cdor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.119 CeilingTruncation Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for CeilingTruncation:



### 7.119.1 Detailed Description

Ceiling truncation.

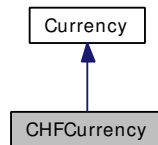
#### Public Member Functions

- `CeilingTruncation` ([Integer](#) precision, [Integer](#) digit=5)

## 7.120 CHFCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for CHFCurrency:



### 7.120.1 Detailed Description

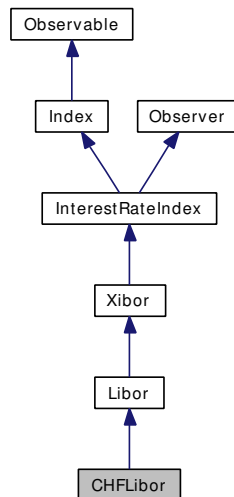
Swiss franc.

The ISO three-letter code is CHF; the numeric code is 756. It is divided into 100 cents.

## 7.121 CHFLibor Class Reference

```
#include <ql/Indexes/chflibor.hpp>
```

Inheritance diagram for CHFLibor:



### 7.121.1 Detailed Description

CHF LIBOR rate

Swiss Franc LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

#### Warning

This is the rate fixed in London by BBA. Use ZIBOR if you're interested in the Zurich fixing.

### Public Member Functions

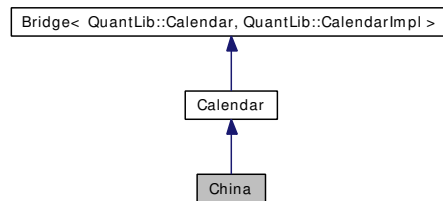
- **CHFLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference, [Integer](#) settlementDays=2)



## 7.122 China Class Reference

```
#include <ql/Calendars/china.hpp>
```

Inheritance diagram for China:



### 7.122.1 Detailed Description

Chinese calendar.

Holidays:

- Saturdays
- Sundays
- New Year's day, January 1st
- Labour Day, first week in May
- National Day, one week from October 1st

Other holidays for which no rule is given:

- Lunar New Year (data available for 2004 only)
- Spring Festival
- Last day of Lunar Year

## 7.123 CLGaussianRng Class Template Reference

```
#include <ql/RandomNumbers/centrallimitgaussianrng.hpp>
```

### 7.123.1 Detailed Description

```
template<class RNG> class QuantLib::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 [Sample< Real >](#) `sample_type`
- typedef RNG `urng_type`

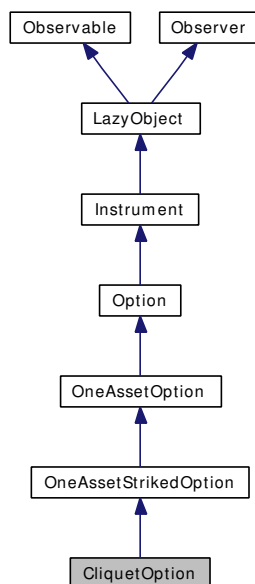
### Public Member Functions

- **CLGaussianRng** (const RNG &uniformGenerator)
- [sample\\_type next](#) () const  
*returns a sample from a Gaussian distribution*

## 7.124 CliquetOption Class Reference

```
#include <ql/Instruments/cliquetoption.hpp>
```

Inheritance diagram for CliquetOption:



### 7.124.1 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.

#### Todo

- add local/global caps/floors
- add accrued coupon and last fixing

### Public Member Functions

- **CliquetOption** (const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [PercentageStrikePayoff](#) > &, const boost::shared\_ptr< [EuropeanExercise](#) > & maturity, const std::vector< [Date](#) > &resetDates, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) \*) const

### Classes

- class [arguments](#)  
*Arguments for cliquet option calculation*

- class [engine](#)

*Cliquet engine base class.*

## 7.124.2 Member Function Documentation

### 7.124.2.1 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 [OneAssetStrikedOption](#).

## 7.125 CliquetOption::arguments Class Reference

```
#include <ql/Instruments/cliquetoption.hpp>
```

### 7.125.1 Detailed Description

Arguments for cliquet option calculation

#### Public Member Functions

- void **validate** () const

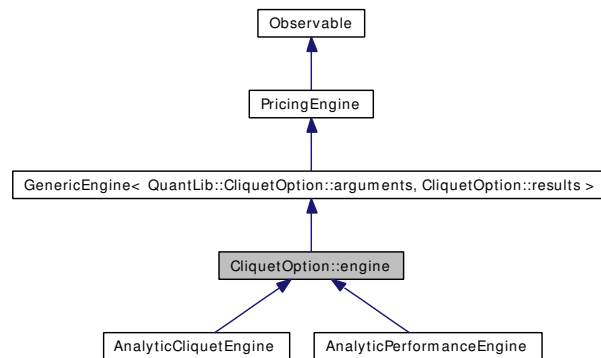
#### Public Attributes

- Real **accruedCoupon**
- Real **lastFixing**
- Real **localCap**
- Real **localFloor**
- Real **globalCap**
- Real **globalFloor**
- std::vector< [Date](#) > **resetDates**

## 7.126 CliquetOption::engine Class Reference

```
#include <ql/Instruments/cliquetoption.hpp>
```

Inheritance diagram for CliquetOption::engine:



### 7.126.1 Detailed Description

Cliquet engine base class.

## 7.127 Clone Class Template Reference

```
#include <ql/Utilities/clone.hpp>
```

### 7.127.1 Detailed Description

**template<class T> class QuantLib::Clone< T >**

cloning proxy to an underlying object

When copied, this class will make a clone of its underlying object (which must provide a `clone()` method returning a `std::auto_ptr` to a newly-allocated instance.)

### Public Member Functions

- **Clone** (`std::auto_ptr< T >`)
- **Clone** (`const T &`)
- **Clone** (`const Clone< T > &`)
- **Clone**< T > & **operator=** (`const T &`)
- **Clone**< T > & **operator=** (`const Clone< T > &`)
- `T & operator *` (`() const`)
- `T * operator →` (`() const`)
- `bool empty` (`() const`)
- `void swap` (`Clone< T > &t`)

### Related Functions

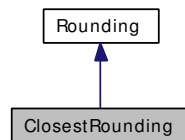
(Note that these are not member functions.)

- `void swap` (`Clone< T > &`, `Clone< T > &`)

## 7.128 ClosestRounding Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for ClosestRounding:



### 7.128.1 Detailed Description

Closest rounding.

#### Public Member Functions

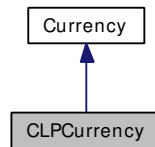
- `ClosestRounding` ([Integer](#) precision, [Integer](#) digit=5)



## 7.129 CLPCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for CLPCurrency:



### 7.129.1 Detailed Description

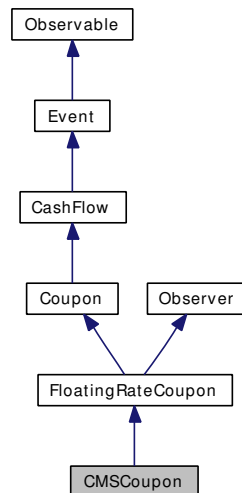
Chilean peso.

The ISO three-letter code is CLP; the numeric code is 152. It is divided in 100 centavos.

## 7.130 CMSCoupon Class Reference

```
#include <ql/CashFlows/cmscoupon.hpp>
```

Inheritance diagram for CMSCoupon:



### 7.130.1 Detailed Description

CMS 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

- **CMSCoupon** (const Real nominal, const [Date](#) &paymentDate, const boost::shared\_ptr< [SwapIndex](#) > &index, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, const [DayCounter](#) &dayCounter, const boost::shared\_ptr< [VanillaCMSCouponPricer](#) > &pricer, Real gearing, Rate spread, Rate cap=[Null](#)< Rate >(), Rate floor=[Null](#)< Rate >(), Real meanReversion=0., const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), bool isInArrears=false)

#### Coupon interface

- Real **price** (const [Handle](#)< [YieldTermStructure](#) > &discountingCurve) const
- Rate **rate** () const  
*accrued rate*
- Rate **rate1** () const

#### Inspectors

- const boost::shared\_ptr< [SwapIndex](#) > & **swapIndex** () const

- Rate **cap** () const
- Rate **floor** () const
- Real **meanReversion** () const
- virtual **Date fixingDate** () const  
*fixing date*

### Modifiers

- void **setSwaptionVolatility** (const [Handle< SwaptionVolatilityStructure >](#) &)
- [Handle< SwaptionVolatilityStructure >](#) **swaptionVolatility** () const

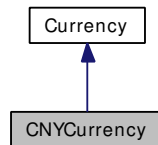
### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.131 CNYCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for CNYCurrency:



### 7.131.1 Detailed Description

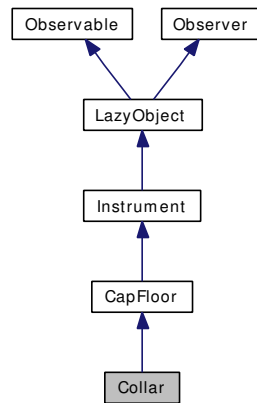
Chinese yuan.

The ISO three-letter code is CNY; the numeric code is 156. It is divided in 100 fen.

## 7.132 Collar Class Reference

```
#include <ql/Instruments/capfloor.hpp>
```

Inheritance diagram for Collar:



### 7.132.1 Detailed Description

Concrete collar class.

#### Public Member Functions

- **Collar** (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &floatingLeg, const std::vector< [Rate](#) > &capRates, const std::vector< [Rate](#) > &floorRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared\_ptr< [PricingEngine](#) > &engine)

## 7.133 Composite Class Template Reference

```
#include <ql/Patterns/composite.hpp>
```

### 7.133.1 Detailed Description

**template<class T> class QuantLib::Composite< T >**

Composite pattern.

The typical use of this class is:

```
class CompositeFoo : public Composite<Foo> {  
    ...  
};
```

which causes CompositeFoo to inherit from Foo and provides it with methods for adding components. Of course, any abstract Foo interface must still be implemented.

### Protected Types

- typedef std::list< boost::shared\_ptr< T > >::iterator **iterator**
- typedef std::list< boost::shared\_ptr< T > >::const\_iterator **const\_iterator**

### Protected Member Functions

- void **add** (const boost::shared\_ptr< T > &c)

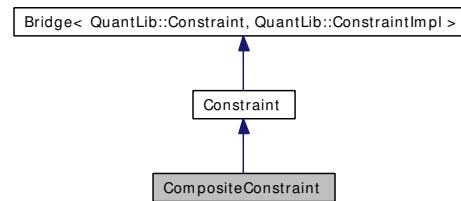
### Protected Attributes

- std::list< boost::shared\_ptr< T > > **components\_**

## 7.134 CompositeConstraint Class Reference

```
#include <ql/Optimization/constraint.hpp>
```

Inheritance diagram for CompositeConstraint:



### 7.134.1 Detailed Description

Constraint enforcing both given sub-constraints

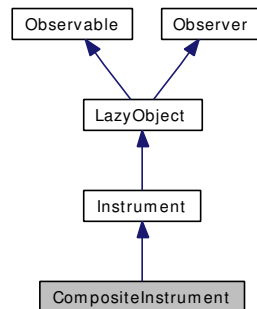
#### Public Member Functions

- `CompositeConstraint` (const [Constraint](#) &c1, const [Constraint](#) &c2)

## 7.135 CompositeInstrument Class Reference

```
#include <ql/Instruments/compositeinstrument.hpp>
```

Inheritance diagram for CompositeInstrument:



### 7.135.1 Detailed Description

**Composite** instrument.

This instrument is an aggregate of other instruments. Its NPV is the sum of the NPVs of its components, each possibly multiplied by a given factor.

**Example:** [static replication of a down-and-out barrier option](#)

#### Warning

Methods that drive the calculation directly (such as [recalculate\(\)](#), [freeze\(\)](#) and others) might not work correctly.

**Examples:**

[Replication.cpp](#).

### Instrument interface

- `bool isExpired () const`  
*returns whether the instrument is still tradable.*
- `void performCalculations () const`

### Public Member Functions

- `void add (const boost::shared_ptr< Instrument > &instrument, Real multiplier=1.0)`  
*adds an instrument to the composite*
- `void subtract (const boost::shared_ptr< Instrument > &instrument, Real multiplier=1.0)`  
*shorts an instrument from the composite*



## 7.135.2 Member Function Documentation

### 7.135.2.1 void performCalculations () const [protected, virtual]

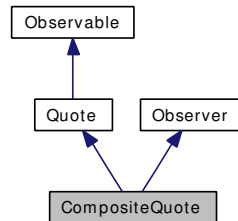
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

## 7.136 CompositeQuote Class Template Reference

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

Inheritance diagram for CompositeQuote:



### 7.136.1 Detailed Description

```
template<class BinaryFunction> class QuantLib::CompositeQuote< BinaryFunction >
```

market element whose value depends on two other market element

#### Tests

the correctness of the returned values is tested by checking them against numerical calculations.

### Public Member Functions

- **CompositeQuote** (const [Handle](#)< [Quote](#) > &element1, const [Handle](#)< [Quote](#) > &element2, const BinaryFunction &f)

#### Quote interface

- [Real value](#) () const  
*returns the current value*

#### Observer interface

- void [update](#) ()

### 7.136.2 Member Function Documentation

#### 7.136.2.1 void update () [virtual]

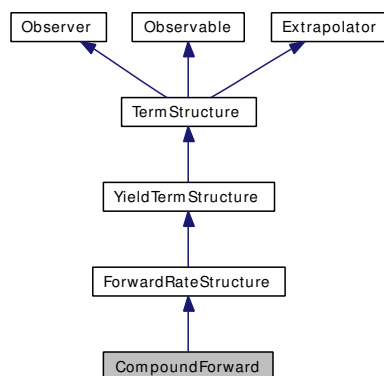
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.137 CompoundForward Class Reference

```
#include <ql/TermStructures/compoundforward.hpp>
```

Inheritance diagram for CompoundForward:



### 7.137.1 Detailed Description

compound-forward structure

#### Tests

- the correctness of the curve is tested by reproducing the input data.
- the correctness of the curve is tested by checking the consistency between returned rates and swaps priced on the curve.

#### Bug

swap rates are not reproduced exactly when using indexed coupons. Apparently, some assumption about the swap fixings is hard-coded into the bootstrapping algorithm.

### Public Member Functions

- **CompoundForward** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &dates, const std::vector< [Rate](#) > &forwards, const [Calendar](#) &calendar, const [BusinessDayConvention](#) conv, const Integer compounding, const [DayCounter](#) &dayCounter)
- [Calendar](#) **calendar** () const  
*the calendar used for reference date calculation*
- [BusinessDayConvention](#) **businessDayConvention** () const
- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*
- Integer **compounding** () const
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*

- Time [maxTime](#) () const  
*the latest time for which the curve can return values*
- const std::vector< Time > & **times** () const
- const std::vector< [Date](#) > & **dates** () const
- const std::vector< Rate > & **forwards** () const
- boost::shared\_ptr< [ExtendedDiscountCurve](#) > **discountCurve** () const
- Rate **compoundForward** (const [Date](#) &d1, Integer f, bool extrapolate=false) const
- Rate **compoundForward** (Time t1, Integer f, bool extrapolate=false) const

## Protected Member Functions

- void **calibrateNodes** () const
- boost::shared\_ptr< [YieldTermStructure](#) > **bootstrap** () const
- Rate [zeroYieldImpl](#) (Time) const
- [DiscountFactor](#) [discountImpl](#) (Time) const
- Size **referenceNode** (Time) const
- Rate [forwardImpl](#) (Time) const  
*instantaneous forward-rate calculation*
- Rate **compoundForwardImpl** (Time, Integer) const

## 7.137.2 Member Function Documentation

### 7.137.2.1 Rate [zeroYieldImpl](#) (Time) 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 and possibly wildly inaccurate implementation. Derived classes should implement their own `zeroYield` method.

Reimplemented from [ForwardRateStructure](#).

### 7.137.2.2 [DiscountFactor](#) [discountImpl](#) (Time) const [protected, virtual]

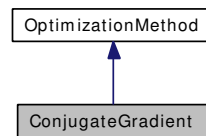
Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Reimplemented from [ForwardRateStructure](#).

## 7.138 ConjugateGradient Class Reference

```
#include <ql/Optimization/conjugategradient.hpp>
```

Inheritance diagram for ConjugateGradient:



### 7.138.1 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](#) (const boost::shared\_ptr< [LineSearch](#) > &lineSearch=boost::shared\_ptr< [LineSearch](#) >())  
*default constructor*
- [ConjugateGradient](#) (const [EndCriteria](#) &endCriteria, const [Array](#) &initialValue, const boost::shared\_ptr< [LineSearch](#) > &lineSearch=boost::shared\_ptr< [LineSearch](#) >())
- void [minimize](#) (const [Problem](#) &P) const  
*minimize the optimization problem P*

## 7.139 ConstantEstimator Class Reference

```
#include <ql/VolatilityModels/constantestimator.hpp>
```

### 7.139.1 Detailed Description

This class implements a concrete volatility model

Volatilities are assumed to be expressed on an annual basis.

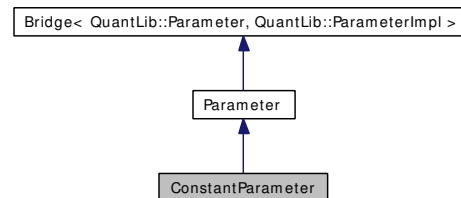
### Public Member Functions

- **ConstantEstimator** (Size size)
- [TimeSeries](#)< Volatility > **calculate** (const [TimeSeries](#)< Volatility > &)
- void **calibrate** (const [TimeSeries](#)< Volatility > &)

## 7.140 ConstantParameter Class Reference

```
#include <ql/ShortRateModels/parameter.hpp>
```

Inheritance diagram for ConstantParameter:



### 7.140.1 Detailed Description

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

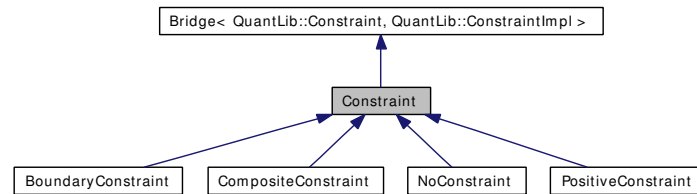
#### Public Member Functions

- **ConstantParameter** (const [Constraint](#) &constraint)
- **ConstantParameter** ([Real](#) value, const [Constraint](#) &constraint)

## 7.141 Constraint Class Reference

```
#include <ql/Optimization/constraint.hpp>
```

Inheritance diagram for Constraint:



### 7.141.1 Detailed Description

Base constraint class.

#### Public Member Functions

- `bool test (const Array &p) const`
- `Real update (Array &p, const Array &direction, Real beta)`
- `Constraint (const boost::shared_ptr< ConstraintImpl > &impl=boost::shared_ptr< ConstraintImpl >())`



## 7.142 ConstraintImpl Class Reference

```
#include <ql/Optimization/constraint.hpp>
```

### 7.142.1 Detailed Description

Base class for constraint implementations.

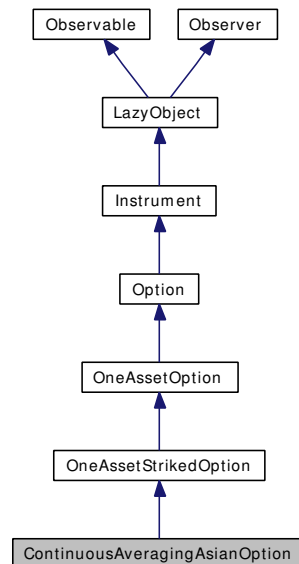
#### Public Member Functions

- virtual bool [test](#) (const [Array](#) &params) const =0  
*Tests if params satisfy the constraint.*

## 7.143 ContinuousAveragingAsianOption Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

Inheritance diagram for ContinuousAveragingAsianOption:



### 7.143.1 Detailed Description

Continuous-averaging Asian option.

#### Todo

add running average

### Public Member Functions

- **ContinuousAveragingAsianOption** (Average::Type averageType, const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) \*) const

### Protected Attributes

- Average::Type [averageType\\_](#)

### Classes

- class [arguments](#)

*Extra arguments for single-asset continuous-average Asian option.*

- class [engine](#)

*Continuous-averaging Asian engine base class.*

## 7.143.2 Member Function Documentation

### 7.143.2.1 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 [OneAssetStrikedOption](#).

## 7.144 ContinuousAveragingAsianOption::arguments Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

### 7.144.1 Detailed Description

Extra arguments for single-asset continuous-average Asian option.

#### Public Member Functions

- void **validate** () const

#### Public Attributes

- Average::Type **averageType**

## 7.145 ContinuousAveragingAsianOption::engine Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

Inheritance diagram for ContinuousAveragingAsianOption::engine:



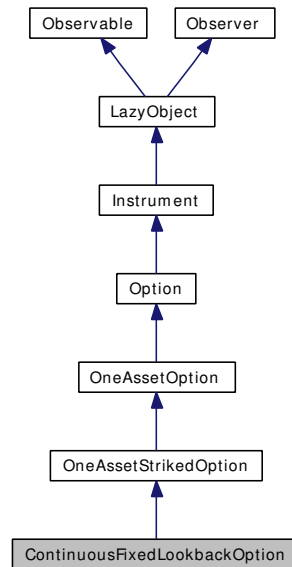
### 7.145.1 Detailed Description

Continuous-averaging Asian engine base class.

## 7.146 ContinuousFixedLookbackOption Class Reference

#include <ql/Instruments/lookbackoption.hpp>

Inheritance diagram for ContinuousFixedLookbackOption:



### 7.146.1 Detailed Description

Continuous-fixed lookback option.

#### Public Member Functions

- **ContinuousFixedLookbackOption** (const Real currentMinmax, const boost::shared\_ptr< [StochasticProcess](#) > &process, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) > ())
- void [setupArguments](#) ([Arguments](#) \*) const

#### Protected Attributes

- Real [minmax\\_](#)

#### Classes

- class [arguments](#)  
*Arguments for continuous fixed lookback option calculation*
- class [engine](#)  
*Continuous fixed lookback engine base class*

## 7.146.2 Member Function Documentation

### 7.146.2.1 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 [OneAssetStrikedOption](#).

## 7.147 ContinuousFixedLookbackOption::arguments Class Reference

```
#include <ql/Instruments/lookbackoption.hpp>
```

### 7.147.1 Detailed Description

Arguments for continuous fixed lookback option calculation

#### Public Member Functions

- void **validate** () const

#### Public Attributes

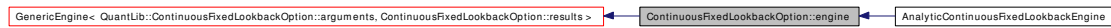
- Real **minmax**



## 7.148 ContinuousFixedLookbackOption::engine Class Reference

```
#include <ql/Instruments/lookbackoption.hpp>
```

Inheritance diagram for ContinuousFixedLookbackOption::engine:



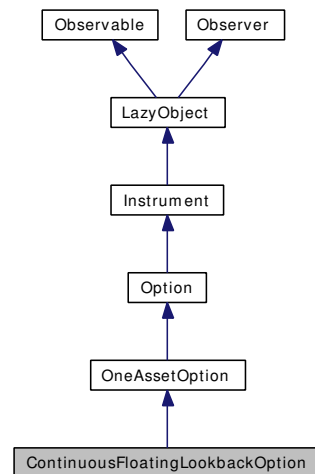
### 7.148.1 Detailed Description

Continuous fixed lookback engine base class

## 7.149 ContinuousFloatingLookbackOption Class Reference

```
#include <ql/Instruments/lookbackoption.hpp>
```

Inheritance diagram for ContinuousFloatingLookbackOption:



### 7.149.1 Detailed Description

Continuous-floating lookback option.

#### Public Member Functions

- **ContinuousFloatingLookbackOption** (const Real currentMinmax, const boost::shared\_ptr< [StochasticProcess](#) > &process, const boost::shared\_ptr< [TypePayoff](#) > &payoff, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) \*) const

#### Protected Attributes

- Real **minmax\_**

#### Classes

- class [arguments](#)  
*Arguments for continuous floating lookback option calculation*
- class [engine](#)  
*Continuous floating lookback engine base class*

## 7.149.2 Member Function Documentation

### 7.149.2.1 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 [OneAssetOption](#).

## 7.150 ContinuousFloatingLookbackOption::arguments Class Reference

```
#include <ql/Instruments/lookbackoption.hpp>
```

### 7.150.1 Detailed Description

Arguments for continuous floating lookback option calculation

#### Public Member Functions

- void **validate** () const

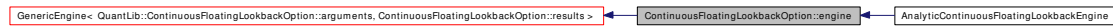
#### Public Attributes

- Real **minmax**

## 7.151 ContinuousFloatingLookbackOption::engine Class Reference

```
#include <ql/Instruments/lookbackoption.hpp>
```

Inheritance diagram for ContinuousFloatingLookbackOption::engine:



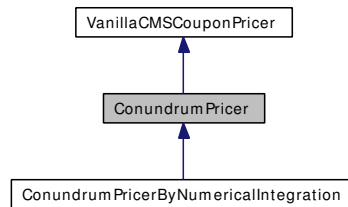
### 7.151.1 Detailed Description

Continuous floating lookback engine base class

## 7.152 ConundrumPricer Class Reference

```
#include <ql/CashFlows/conundrumpricer.hpp>
```

Inheritance diagram for ConundrumPricer:



### 7.152.1 Detailed Description

[ConundrumPricer](#).

Base class for the pricing of a CMS coupon via static replication as in Hagan's "Conundrums..." article

#### Public Member Functions

- Real **price** () const
- Real **rate** () const

#### Protected Member Functions

- **ConundrumPricer** (const GFunctionFactory::ModelOfYieldCurve modelOfYieldCurve)
- void **initialize** (const [CMSCoupon](#) &coupon)
- virtual Real **optionLetPrice** (Option::Type optionType, Real strike) const=0
- virtual Real **swapLetPrice** () const=0

#### Protected Attributes

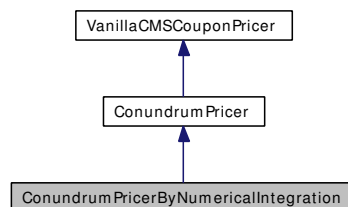
- boost::shared\_ptr< [YieldTermStructure](#) > **rateCurve\_**
- GFunctionFactory::ModelOfYieldCurve **modelOfYieldCurve\_**
- boost::shared\_ptr< GFunction > **gFunction\_**
- const [CMSCoupon](#) \* **coupon\_**
- [Date](#) **paymentDate\_**
- [Date](#) **fixingDate\_**
- Real **swapRateValue\_**
- Real **discount\_**
- Real **annuity\_**
- Real **min\_**
- Real **max\_**
- Real **gearing\_**
- Real **spread\_**
- const Real **cutoffForCaplet\_**

- const Real **cutoffForFloorlet\_**
- [Period](#) **swapTenor\_**
- boost::shared\_ptr< VanillaOptionPricer > **vanillaOptionPricer\_**

## 7.153 ConundrumPricerByNumericalIntegration Class Reference

```
#include <ql/CashFlows/conundrumpricer.hpp>
```

Inheritance diagram for ConundrumPricerByNumericalIntegration:



### 7.153.1 Detailed Description

[ConundrumPricerByNumericalIntegration](#).

Prices a CMS coupon via static replication as in Hagan's "Conundrums..." article via numerical integration based on prices of vanilla swaptions

### Public Member Functions

- **ConundrumPricerByNumericalIntegration** (const GFunctionFactory::ModelOfYieldCurve modelOfYieldCurve=GFunctionFactory::standard, Real lowerLimit=0.0, Real upperLimit=1.0)



## 7.154 ConvergenceStatistics Class Template Reference

```
#include <ql/Math/convergencestatistics.hpp>
```

### 7.154.1 Detailed Description

**template<class T, class U = DoublingConvergenceSteps> class QuantLib::ConvergenceStatistics< T, U >**

statistics class with convergence table

This class decorates another statistics class adding a convergence table calculation. The table tracks the convergence of the mean.

It is possible to specify the number of samples at which the mean should be stored by mean of the second template parameter; the default is to store  $2^{n-1}$  samples at the  $n$ -th step. Any passed class must implement the following interface:

```
Size initialSamples() const;
Size nextSamples(Size currentSamples) const;
```

as well as a copy constructor.

#### Tests

results are tested against known good values.

### Public Types

- typedef T::value\_type **value\_type**
- typedef std::vector< std::pair< Size, value\_type > > **table\_type**

### Public Member Functions

- **ConvergenceStatistics** (const T &stats, const U &rule=U())
- **ConvergenceStatistics** (const U &rule=U())
- void **add** (const value\_type &value, [Real](#) 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)
- void **reset** ()
- const std::vector< std::pair< Size, value\_type > > & **convergenceTable** () const

## 7.155 ConvertibleBond::option::arguments Class Reference

```
#include <ql/Instruments/convertiblebond.hpp>
```

### 7.155.1 Detailed Description

Arguments for Convertible [Bond](#) calculation

#### Public Member Functions

- void **validate** () const

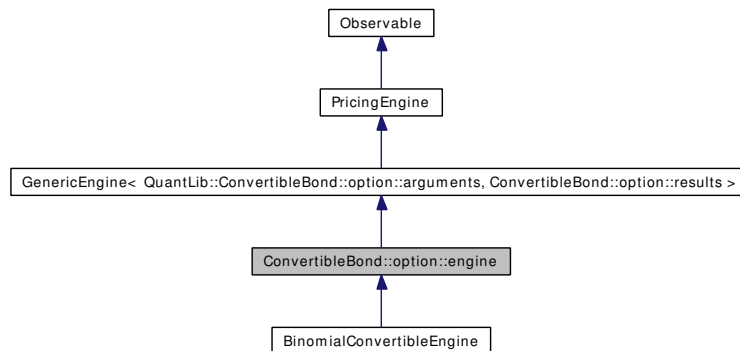
#### Public Attributes

- Real **conversionRatio**
- [Handle](#)< [Quote](#) > **creditSpread**
- DividendSchedule **dividends**
- std::vector< Time > **dividendTimes**
- std::vector< Time > **callabilityTimes**
- std::vector< [Callability::Type](#) > **callabilityTypes**
- std::vector< Real > **callabilityPrices**
- std::vector< Real > **callabilityTriggers**
- std::vector< Time > **couponTimes**
- std::vector< Real > **couponAmounts**
- [DayCounter](#) **dayCounter**
- [Date](#) **issueDate**
- [Date](#) **settlementDate**
- Integer **settlementDays**
- Real **redemption**

## 7.156 ConvertibleBond::option::engine Class Reference

```
#include <ql/Instruments/convertiblebond.hpp>
```

Inheritance diagram for ConvertibleBond::option::engine:



### 7.156.1 Detailed Description

convertible bond engine base class

## 7.157 ConvertibleFixedCouponBond Class Reference

```
#include <ql/Instruments/convertiblebond.hpp>
```

### 7.157.1 Detailed Description

convertible fixed-coupon bond

#### Warning

Most methods inherited from [Bond](#) (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

#### Examples:

[ConvertibleBonds.cpp](#).

### Public Member Functions

- **ConvertibleFixedCouponBond** (const boost::shared\_ptr< [StochasticProcess](#) > &process, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine, [Real](#) conversionRatio, const DividendSchedule &dividends, const CallabilitySchedule &callability, const [Handle](#)< [Quote](#) > &creditSpread, const [Date](#) &issueDate, [Integer](#) settlementDays, const std::vector< [Rate](#) > &coupons, const [DayCounter](#) &dayCounter, const [Schedule](#) &schedule, [Real](#) redemption=100)

## 7.158 ConvertibleFloatingRateBond Class Reference

```
#include <ql/Instruments/convertiblebond.hpp>
```

### 7.158.1 Detailed Description

convertible floating-rate bond

#### Warning

Most methods inherited from [Bond](#) (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

### Public Member Functions

- **ConvertibleFloatingRateBond** (const boost::shared\_ptr< [StochasticProcess](#) > &process, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine, [Real](#) conversionRatio, const DividendSchedule &dividends, const CallabilitySchedule &callability, const [Handle](#)< [Quote](#) > &creditSpread, const [Date](#) &issueDate, [Integer](#) settlementDays, const boost::shared\_ptr< [Xibor](#) > &index, [Integer](#) fixingDays, const std::vector< [Spread](#) > &spreads, const [DayCounter](#) &dayCounter, const [Schedule](#) &schedule, [Real](#) redemption=100)

## 7.159 ConvertibleZeroCouponBond Class Reference

```
#include <ql/Instruments/convertiblebond.hpp>
```

### 7.159.1 Detailed Description

convertible zero-coupon bond

#### Warning

Most methods inherited from [Bond](#) (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

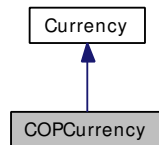
### Public Member Functions

- **ConvertibleZeroCouponBond** (const boost::shared\_ptr< [StochasticProcess](#) > &process, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine, [Real](#) conversionRatio, const DividendSchedule &dividends, const CallabilitySchedule &callability, const [Handle](#)< [Quote](#) > &creditSpread, const [Date](#) &issueDate, [Integer](#) settlementDays, const [DayCounter](#) &dayCounter, const [Schedule](#) &schedule, [Real](#) redemption=100)

## 7.160 COPCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for COPCurrency:



### 7.160.1 Detailed Description

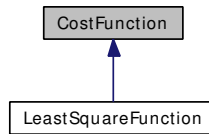
Colombian peso.

The ISO three-letter code is COP; the numeric code is 170. It is divided in 100 centavos.

## 7.161 CostFunction Class Reference

```
#include <ql/Optimization/costfunction.hpp>
```

Inheritance diagram for CostFunction:



### 7.161.1 Detailed Description

Cost function abstract class for optimization problem.

#### Public Member Functions

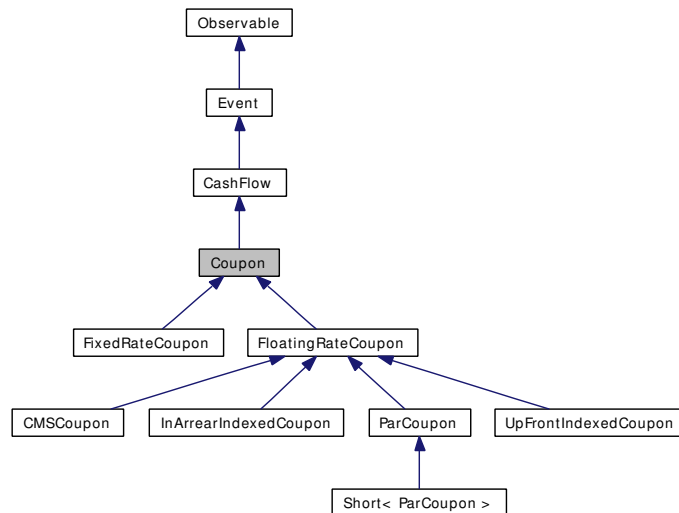
- virtual [Real value](#) (const [Array](#) &x) const=0  
*method to overload to compute the cost functon value in x*
- virtual [Disposable< Array > values](#) (const [Array](#) &x) const  
*const function value for least square optimization*
- virtual void [gradient](#) ([Array](#) &grad, const [Array](#) &x) const  
*method to overload to compute grad\_f, the first derivative of*
- virtual [Real valueAndGradient](#) ([Array](#) &grad, const [Array](#) &x) const  
*method to overload to compute grad\_f, the first derivative of*
- virtual [Real finiteDifferenceEpsilon](#) () const  
*Default epsilon for finite difference method .:*



## 7.162 Coupon Class Reference

```
#include <ql/CashFlows/coupon.hpp>
```

Inheritance diagram for Coupon:



### 7.162.1 Detailed Description

coupon accruing over a fixed period

This class implements part of the [CashFlow](#) interface but it is still abstract and provides derived classes with methods for accrual period calculations.

### Public Member Functions

- [Coupon](#) (Real nominal, const [Date](#) &paymentDate, const [Date](#) &accrualStartDate, const [Date](#) &accrualEndDate, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)())

### Partial CashFlow interface

- [Date](#) date () const

*Note:*

*This is inherited from the event class*

### Inspectors

- Real **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*
- [Integer accrualDays](#) () const  
*accrual period in days*
- virtual [Rate rate](#) () const=0  
*accrued rate*
- virtual [DayCounter dayCounter](#) () const=0  
*day counter for accrual calculation*
- virtual Real [accruedAmount](#) (const [Date](#) &) const =0  
*accrued amount at the given date*

### Visitability

- virtual void [accept](#) ([AcyclicVisitor](#) &)

### Protected Attributes

- Real [nominal\\_](#)
- [Date](#) [paymentDate\\_](#)
- [Date](#) [accrualStartDate\\_](#)
- [Date](#) [accrualEndDate\\_](#)
- [Date](#) [refPeriodStart\\_](#)
- [Date](#) [refPeriodEnd\\_](#)

## 7.162.2 Constructor & Destructor Documentation

- 7.162.2.1 [Coupon](#) (Real *nominal*, const [Date](#) & *paymentDate*, const [Date](#) & *accrualStartDate*, const [Date](#) & *accrualEndDate*, const [Date](#) & *refPeriodStart* = [Date](#)(), const [Date](#) & *refPeriodEnd* = [Date](#)())

### Warning

the coupon does not adjust the payment date which must already be a business day.

## 7.163 CovarianceDecomposition Class Reference

```
#include <ql/MonteCarlo/getcovariance.hpp>
```

### 7.163.1 Detailed Description

Extracts the correlation matrix and the vector of volatilities out of the input covariance matrix. Note that only the lower symmetric part of the covariance matrix is used.

#### Precondition:

The covariance matrix must be symmetric.

#### Tests

cross checked with getCovariance

### Public Member Functions

- [CovarianceDecomposition](#) (const [Matrix](#) &covarianceMatrix, [Real](#) tolerance=1.0e-12)
- const [Array](#) & [variances](#) () const
- const [Array](#) & [standardDeviations](#) () const
- const [Matrix](#) & [correlationMatrix](#) () const

### 7.163.2 Constructor & Destructor Documentation

**7.163.2.1** [CovarianceDecomposition](#) (const [Matrix](#) & *covarianceMatrix*, [Real](#) *tolerance* = 1.0e-12)

#### Precondition:

*covarianceMatrix* must be symmetric

### 7.163.3 Member Function Documentation

**7.163.3.1** const [Array](#)& [variances](#) () const

returns the variances [Array](#)

**7.163.3.2** const [Array](#)& [standardDeviations](#) () const

returns the standard deviations [Array](#)

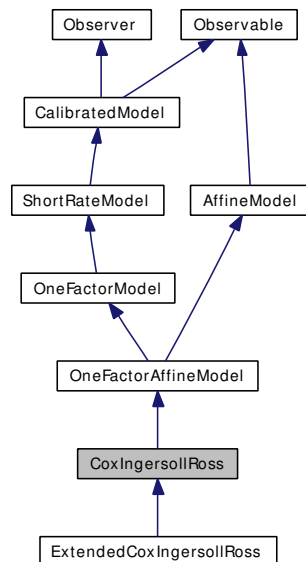
**7.163.3.3** const [Matrix](#)& [correlationMatrix](#) () const

returns the correlation matrix

## 7.164 CoxIngersollRoss Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/coxingersollross.hpp>
```

Inheritance diagram for CoxIngersollRoss:



### 7.164.1 Detailed Description

Cox-Ingersoll-Ross model class.

This class implements the Cox-Ingersoll-Ross model defined by

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

#### Bug

this class was not tested enough to guarantee its functionality.

### Public Member Functions

- **CoxIngersollRoss** ([Rate](#) r0=0.05, Real theta=0.1, Real k=0.1, Real sigma=0.1)
- virtual Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const
- virtual boost::shared\_ptr< ShortRateDynamics > [dynamics](#) () const  
*returns the short-rate dynamics*
- boost::shared\_ptr< [NumericalMethod](#) > [tree](#) (const [TimeGrid](#) &grid) const  
*Return by default a trinomial recombining tree.*

## Protected Member Functions

- Real **A** (Time t, Time T) const
- Real **B** (Time t, Time T) const
- Real **theta** () const
- Real **k** () const
- Real **sigma** () const
- Real **x0** () const

## Classes

- class [Dynamics](#)  
*Dynamics of the short-rate under the Cox-Ingersoll-Ross model*

## 7.165 CoxIngersollRoss::Dynamics Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/coxingersollross.hpp>
```

### 7.165.1 Detailed Description

Dynamics 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

$$dy_t = \left[ \left( \frac{k\theta}{2} + \frac{\sigma^2}{8} \right) \frac{1}{y_t} - \frac{k}{2} y_t \right] dt + \frac{\sigma}{2} dW_t$$

.

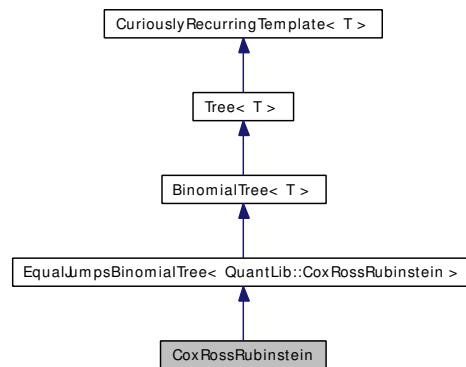
### Public Member Functions

- **Dynamics** ([Real](#) theta, [Real](#) k, [Real](#) sigma, [Real](#) x0)
- virtual [Real](#) **variable** ([Time](#), [Rate](#) r) const
- virtual [Real](#) **shortRate** ([Time](#), [Real](#) y) const

## 7.166 CoxRossRubinstein Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for CoxRossRubinstein:



### 7.166.1 Detailed Description

Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.

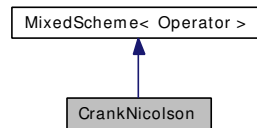
#### Public Member Functions

- **CoxRossRubinstein** (const boost::shared\_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)

## 7.167 CrankNicolson Class Template Reference

```
#include <ql/FiniteDifferences/cranknicolson.hpp>
```

Inheritance diagram for CrankNicolson:



### 7.167.1 Detailed Description

```
template<class Operator> class QuantLib::CrankNicolson< Operator >
```

Crank-Nicolson scheme for finite difference methods.

In this implementation, the passed operator must be derived from either `TimeConstantOperator` or `TimeDependentOperator`. Also, it must implement at least the following interface:

```

typedef ... array_type;

// 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
array_type applyTo(const array_type&);
array_type solveFor(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, 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.

### Public Types

- `typedef OperatorTraits< Operator > traits`
- `typedef traits::operator_type operator_type`
- `typedef traits::array_type array_type`
- `typedef traits::bc_set bc_set`
- `typedef traits::condition_type condition_type`



## Public Member Functions

- **CrankNicolson** (const operator\_type &L, const bc\_set &bcs)

## 7.168 Cubic Class Reference

```
#include <ql/Math/cubicspline.hpp>
```

### 7.168.1 Detailed Description

cubic-spline interpolation factory and traits

#### Public Types

- enum { **global** = 1 }

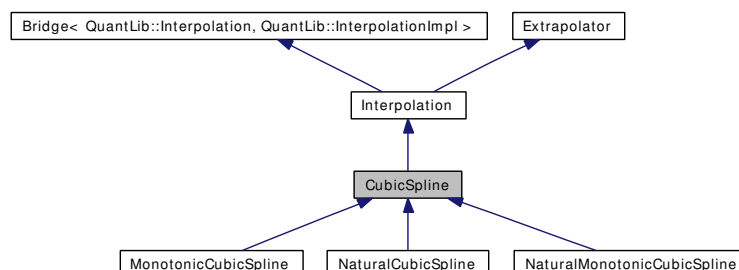
#### Public Member Functions

- **Cubic** ([CubicSpline::BoundaryCondition](#) leftCondition=CubicSpline::SecondDerivative, Real leftConditionValue=0.0, [CubicSpline::BoundaryCondition](#) rightCondition=CubicSpline::SecondDerivative, Real rightConditionValue=0.0, bool monotonicity-Constraint=false)
- template<class I1, class I2> [Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

## 7.169 CubicSpline Class Reference

```
#include <ql/Math/cubicspline.hpp>
```

Inheritance diagram for CubicSpline:



### 7.169.1 Detailed Description

Cubic spline interpolation between discrete points.

It implements different type of end conditions: not-a-knot, first derivative value, second derivative value.

It also implements Hyman's monotonicity constraint filter which ensures that the interpolating spline remains monotonic at the expense of the second derivative of the curve which will no longer be continuous where the filter has been applied. If the interpolating spline is already monotonic, the Hyman filter leaves it unchanged.

See R. L. Dougherty, A. Edelman, and J. M. Hyman, "Nonnegativity-, Monotonicity-, or Convexity-Preserving Cubic and Quintic Hermite Interpolation" Mathematics Of Computation, v. 52, n. 186, April 1989, pp. 471-494.

#### Tests

the correctness of the returned values is tested by reproducing results available in literature.

### Public Types

- enum [BoundaryCondition](#) {  
[NotAKnot](#), [FirstDerivative](#), [SecondDerivative](#), [Periodic](#),  
[Lagrange](#) }

### Public Member Functions

- template<class I1, class I2> [CubicSpline](#) (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, [CubicSpline::BoundaryCondition](#) leftCondition, [Real](#) leftConditionValue, [CubicSpline::BoundaryCondition](#) rightCondition, [Real](#) rightConditionValue, bool monotonicity-Constraint)
- const std::vector< [Real](#) > & [aCoefficients](#) () const
- const std::vector< [Real](#) > & [bCoefficients](#) () const
- const std::vector< [Real](#) > & [cCoefficients](#) () const

## 7.169.2 Member Enumeration Documentation

### 7.169.2.1 enum [BoundaryCondition](#)

#### Enumerator:

*NotAKnot* Make second(-last) point an inactive knot.

*FirstDerivative* Match value of end-slope.

*SecondDerivative* Match value of second derivative at end.

*Periodic* Match first and second derivative at either end.

*Lagrange* Match end-slope to the slope of the cubic that matches the first four data at the respective end

## 7.169.3 Constructor & Destructor Documentation

7.169.3.1 [CubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*,  
[CubicSpline::BoundaryCondition](#) *leftCondition*, [Real](#) *leftConditionValue*,  
[CubicSpline::BoundaryCondition](#) *rightCondition*, [Real](#) *rightConditionValue*, bool  
*monotonicityConstraint*)

#### Precondition:

the *x* values must be sorted.

## 7.170 CumulativeBinomialDistribution Class Reference

```
#include <ql/Math/binomialdistribution.hpp>
```

### 7.170.1 Detailed Description

Cumulative binomial distribution function.

Given an integer  $k$  it provides the cumulative probability of observing  $k \leq k$ : formula here ...

### Public Member Functions

- **CumulativeBinomialDistribution** (Real  $p$ , BigNatural  $n$ )
- Real **operator()** (BigNatural  $k$ ) const

## 7.171 CumulativeNormalDistribution Class Reference

```
#include <ql/Math/normaldistribution.hpp>
```

### 7.171.1 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** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real  $x$ ) const
- Real **derivative** (Real  $x$ ) const

## 7.172 CumulativePoissonDistribution Class Reference

```
#include <ql/Math/poissondistribution.hpp>
```

### 7.172.1 Detailed Description

Cumulative Poisson distribution function.

This function provides an approximation of the integral of the Poisson distribution.

For this implementation see "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery, chapter 6

#### Tests

the correctness of the returned value is tested by checking it against known good results.

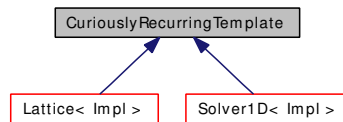
### Public Member Functions

- **CumulativePoissonDistribution** (Real mu)
- Real **operator()** (BigNatural k) const

## 7.173 CuriouslyRecurringTemplate Class Template Reference

```
#include <ql/Patterns/curiouslyrecurring.hpp>
```

Inheritance diagram for CuriouslyRecurringTemplate:



### 7.173.1 Detailed Description

```
template<class Impl> class QuantLib::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

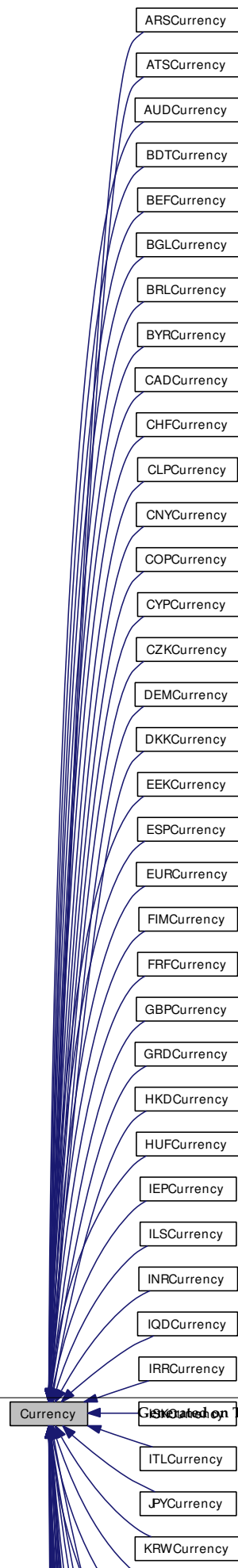
- Impl & **impl** ()
- const Impl & **impl** () const



## 7.174 Currency Class Reference

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

Inheritance diagram for Currency:



### 7.174.1 Detailed Description

Currency specification

#### Public Member Functions

- [Currency](#) ()  
*default constructor*

#### Inspectors

- const std::string & [name](#) () const  
*currency name, e.g, "U.S. Dollar"*
- const std::string & [code](#) () const  
*ISO 4217 three-letter code, e.g, "USD".*
- [Integer numericCode](#) () const  
*ISO 4217 numeric code, e.g, "840".*
- const std::string & [symbol](#) () const  
*symbol, e.g, "\$"*
- const std::string & [fractionSymbol](#) () const  
*fraction symbol, e.g, "¢"*
- [Integer fractionsPerUnit](#) () const  
*number of fractionary parts in a unit, e.g, 100*
- const [Rounding](#) & [rounding](#) () const  
*rounding convention*
- std::string [format](#) () const  
*output format*

#### other info

- bool [isValid](#) () const  
*is this a usable instance?*
- const [Currency](#) & [triangulationCurrency](#) () const  
*currency used for triangulated exchange when required*

#### Protected Attributes

- boost::shared\_ptr< Data > [data\\_](#)

## Related Functions

(Note that these are not member functions.)

- `bool operator==` (const [Currency](#) &, const [Currency](#) &)
- `bool operator!=` (const [Currency](#) &, const [Currency](#) &)
- `std::ostream & operator<<` (std::ostream &, const [Currency](#) &)

## 7.174.2 Constructor & Destructor Documentation

### 7.174.2.1 [Currency](#) ()

default constructor

Instances built via this constructor have undefined behavior. Such instances can only act as placeholders and must be reassigned to a valid currency before being used.

## 7.174.3 Member Function Documentation

### 7.174.3.1 `std::string format () const`

output format

The format will be fed three positional parameters, namely, value, code, and symbol, in this order.

## 7.175 CurveState Class Reference

```
#include <ql/MarketModels/curvestate.hpp>
```

### 7.175.1 Detailed Description

This class stores the state of the yield curve associated to the fixed calendar times within the simulation.

This is the workhorse discounting object associated to the rate times of the simulation. It's important to pass the rates via an object like this to the product rather than directly to make it easier to switch to other engines such as a coterminal-swap-rate engine.

Many products will not need expired rates and others will only require the first rate.

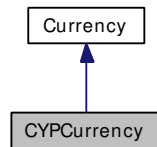
### Public Member Functions

- **CurveState** (const std::vector< Time > &rateTimes)
- template<class ForwardIterator> **CurveState** (ForwardIterator begin, ForwardIterator end)
- const std::vector< Time > & **rateTimes** () const
- void **setOnForwardRates** (const std::vector< Rate > &rates)
- template<class ForwardIterator> void **setOnForwardRates** (ForwardIterator begin, ForwardIterator end)
- void **setOnDiscountRatios** (const std::vector< DiscountFactor > &discountRatios)
- void **setOnCoterminalSwapRates** (const std::vector< Rate > &swapRates)
- const std::vector< Rate > & **forwardRates** () const
- const std::vector< DiscountFactor > & **discountRatios** () const
- const std::vector< Rate > & **coterminalSwapRates** () const
- const std::vector< Real > & **coterminalSwapRatesAnnuities** () const
- Rate **forwardRate** (Size i) const
- Real **discountRatio** (Size i, Size j) const
- Rate **coterminalSwapRate** (Size i) const
- const std::vector< Time > & **rateTaus** () const

## 7.176 CYPCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for CYPCurrency:



### 7.176.1 Detailed Description

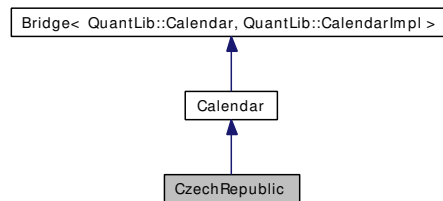
Cyprus pound.

The ISO three-letter code is CYP; the numeric code is 196. It is divided in 100 cents.

## 7.177 CzechRepublic Class Reference

```
#include <ql/Calendars/czechrepublic.hpp>
```

Inheritance diagram for CzechRepublic:



### 7.177.1 Detailed Description

Czech calendars.

Holidays for the Prague stock exchange (see <http://www.pse.cz/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Easter Monday
- Labour Day, May 1st
- Liberation Day, May 8th
- SS. Cyril and Methodius, July 5th
- Jan Hus Day, July 6th
- Czech Statehood Day, September 28th
- Independence Day, October 28th
- Struggle for Freedom and Democracy Day, November 17th
- Christmas Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th

### Public Types

- enum `Market` { `PSE` }

### Public Member Functions

- `CzechRepublic` (`Market` m=PSE)

## 7.177.2 Member Enumeration Documentation

### 7.177.2.1 enum [Market](#)

Enumerator:

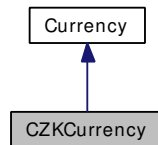
*PSE* Prague stock exchange.



## 7.178 CZKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for CZKCurrency:



### 7.178.1 Detailed Description

Czech koruna.

The ISO three-letter code is CZK; the numeric code is 203. It is divided in 100 haleru.

## 7.179 Date Class Reference

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

### 7.179.1 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).

#### Tests

self-consistency of dates, serial numbers, days of month, months, and weekdays is checked over the whole date range.

#### Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), [FRA.cpp](#), [Replication.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

### Public Member Functions

#### inspectors

- [Weekday](#) **weekday** () const
- Day **dayOfMonth** () const
- Day [dayOfYear](#) () const  
*One-based (Jan 1st = 1).*
- [Month](#) **month** () const
- Year **year** () const
- BigInteger **serialNumber** () const

#### date algebra

- [Date](#) & **operator+=** (BigInteger days)  
*increments date by the given number of days*
- [Date](#) & **operator+=** (const [Period](#) &)  
*increments date by the given period*
- [Date](#) & **operator-=** (BigInteger days)  
*decrement date by the given number of days*
- [Date](#) & **operator-=** (const [Period](#) &)  
*decrements date by the given period*
- [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+](#) (BigInteger days) const  
*returns a new date incremented by the given number of days*
- [Date operator+](#) (const [Period](#) &) const  
*returns a new date incremented by the given period*
- [Date operator-](#) (BigInteger days) const  
*returns a new date decremented by the given number of days*
- [Date operator-](#) (const [Period](#) &) const  
*returns a new date decremented by the given period*

## Static Public Member Functions

### static methods

- static [Date todaysDate](#) ()  
*today's date.*
- static [Date minDate](#) ()  
*earliest allowed date*
- static [Date maxDate](#) ()  
*latest allowed date*
- static bool [isLeap](#) (Year y)  
*whether the given year is a leap one*
- static [Date endOfMonth](#) (const [Date](#) &d)  
*last day of the month to which the given date belongs*
- static bool [isEOM](#) (const [Date](#) &d)  
*whether a date is the last day of its month*
- static [Date nextWeekday](#) (const [Date](#) &d, [Weekday](#))  
*next given weekday following or equal to the given date*
- static [Date nthWeekday](#) (Size n, [Weekday](#), [Month](#) m, Year y)  
*n-th given weekday in the given month and year*
- static bool [isIMMdate](#) (const [Date](#) &d, bool mainCycle=true)  
*whether or not the given date is an IMM date*
- static [Date nextIMMdate](#) (const [Date](#) &d, bool mainCycle=true)  
*next IMM date following (or equal to) the given date*
- static std::string [IMMcode](#) (const [Date](#) &date)
- static [Date IMMdate](#) (const std::string &IMMcode, const [Date](#) &referenceDate=[Date](#)())

## Related Functions

(Note that these are not member functions.)

- `BigInteger 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` &)
- `std::ostream & operator<<` (std::ostream &, const `Date` &)

## 7.179.2 Member Function Documentation

### 7.179.2.1 `static Date nextWeekday (const Date & d, Weekday) [static]`

next given weekday following or equal to the given date

E.g., the Friday following Tuesday, January 15th, 2002 was January 18th, 2002.

see <http://www.cpearson.com/excel/DateTimeWS.htm>

### 7.179.2.2 `static Date nthWeekday (Size n, Weekday, Month m, Year y) [static]`

n-th given weekday in the given month and year

E.g., the 4th Thursday of March, 1998 was March 26th, 1998.

see <http://www.cpearson.com/excel/DateTimeWS.htm>

### 7.179.2.3 `static Date nextIMMdate (const Date & d, bool mainCycle = true) [static]`

next `IMM` date following (or equal to) the given date

returns the 1st delivery date for next contract listed in the International `Money` Market section of the Chicago Mercantile Exchange.

#### Warning

The result date is following or equal to the original date.

### 7.179.2.4 `static std::string IMMcode (const Date & date) [static]`

returns the `IMM` code for the given date (e.g. H6 for March 15th, 2006).

#### Warning

It raise an exception if the input date is not an `IMM` date

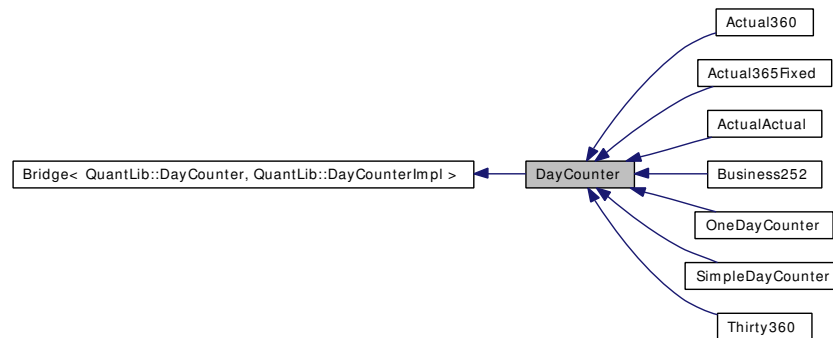
7.179.2.5 static [Date](#) IMMdate (const std::string & *IMMcode*, const [Date](#) & *referenceDate* = [Date](#)()) [static]

returns the [IMM](#) date for the given [IMM](#) code (e.g. March 15th, 2006 for H6).

## 7.180 DayCounter Class Reference

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

Inheritance diagram for DayCounter:



### 7.180.1 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.

**Examples:**

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), [FRA.cpp](#), [Replication.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

### Public Member Functions

- [DayCounter](#) ()

#### DayCounter interface

- `std::string name () const`  
*Returns the name of the day counter.*
- `BigInteger 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 boost::shared\_ptr< [DayCounterImpl](#) > &impl)

## Related Functions

(Note that these are not member functions.)

- `bool operator==(const DayCounter &, const DayCounter &)`
- `bool operator!=(const DayCounter &, const DayCounter &)`
- `std::ostream & operator<< (std::ostream &, const DayCounter &)`

## 7.180.2 Constructor & Destructor Documentation

### 7.180.2.1 DayCounter ()

This default constructor returns a day counter with a null implementation, which is therefore unusable except as a placeholder.

### 7.180.2.2 DayCounter (const boost::shared\_ptr< DayCounterImpl > & impl) [protected]

This protected constructor will only be invoked by derived classes which define a given DayCounter implementation

## 7.180.3 Member Function Documentation

### 7.180.3.1 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.

## 7.180.4 Friends And Related Function Documentation

### 7.180.4.1 bool operator==(const DayCounter &, const DayCounter &) [related]

Returns true iff the two day counters belong to the same derived class.

## 7.181 DayCounterImpl Class Reference

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

### 7.181.1 Detailed Description

abstract base class for day counter implementations

#### Public Member Functions

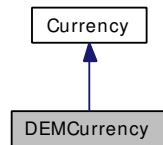
- virtual std::string **name** () const=0
- virtual BigInteger **dayCount** (const [Date](#) &d1, const [Date](#) &d2) const  
*to be overloaded by more complex day counters*
- virtual [Time](#) **yearFraction** (const [Date](#) &, const [Date](#) &, const [Date](#) &refPeriodStart, const [Date](#) &refPeriodEnd) const=0



## 7.182 DEMCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for DEMCurrency:



### 7.182.1 Detailed Description

Deutsche mark.

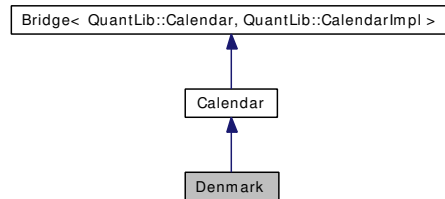
The ISO three-letter code was DEM; the numeric code was 276. It was divided into 100 pfennig.

Obsoleted by the Euro since 1999.

## 7.183 Denmark Class Reference

```
#include <ql/Calendars/denmark.hpp>
```

Inheritance diagram for Denmark:



### 7.183.1 Detailed Description

Danish calendar.

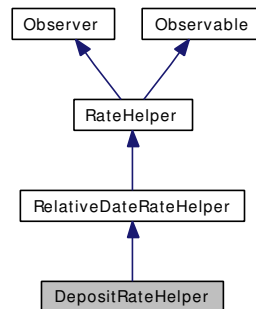
Holidays:

- Saturdays
- Sundays
- Maunday Thursday
- Good Friday
- Easter Monday
- General Prayer Day, 25 days after Easter Monday
- Ascension
- Whit (Pentecost) Monday
- New Year's Day, January 1st
- Constitution Day, June 5th
- Christmas, December 25th
- Boxing Day, December 26th

## 7.184 DepositRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for DepositRateHelper:



### 7.184.1 Detailed Description

Rate helper for bootstrapping over deposit rates.

Examples:

[swapvaluation.cpp](#).

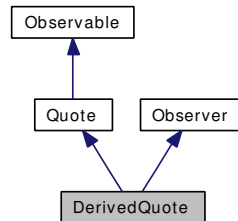
### Public Member Functions

- **DepositRateHelper** (const [Handle](#)< [Quote](#) > &rate, const [Period](#) &tenor, Integer settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **DepositRateHelper** ([Rate](#) rate, const [Period](#) &tenor, Integer settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **Real impliedQuote** () const
- **DiscountFactor discountGuess** () const

## 7.185 DerivedQuote Class Template Reference

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

Inheritance diagram for DerivedQuote:



### 7.185.1 Detailed Description

```
template<class UnaryFunction> class QuantLib::DerivedQuote< UnaryFunction >
```

market element whose value depends on another market element

#### Tests

the correctness of the returned values is tested by checking them against numerical calculations.

### Public Member Functions

- **DerivedQuote** (const [Handle](#)< [Quote](#) > &element, const UnaryFunction &f)

#### Market element interface

- [Real value](#) () const  
*returns the current value*

#### Observer interface

- void [update](#) ()

### 7.185.2 Member Function Documentation

#### 7.185.2.1 void update () [virtual]

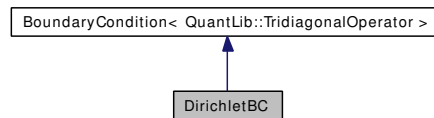
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.186 DirichletBC Class Reference

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Inheritance diagram for DirichletBC:



### 7.186.1 Detailed Description

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

#### Todo

generalize to time-dependent conditions.

### Public Member Functions

- **DirichletBC** (Real value, [Side](#) side)
- void **applyBeforeApplying** ([TridiagonalOperator](#) &) const
- void **applyAfterApplying** ([Array](#) &) const
- void **applyBeforeSolving** ([TridiagonalOperator](#) &, [Array](#) &rhs) const
- void **applyAfterSolving** ([Array](#) &) const
- void **setTime** (Time)

### 7.186.2 Member Function Documentation

#### 7.186.2.1 void setTime (Time) [virtual]

This method sets the current time for time-dependent boundary conditions.

Implements [BoundaryCondition](#).

## 7.187 Discount Struct Reference

```
#include <ql/TermStructures/bootstraptraits.hpp>
```

### 7.187.1 Detailed Description

Discount-curve traits.

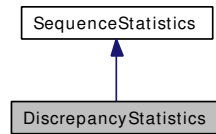
#### Static Public Member Functions

- static [DiscountFactor](#) **initialValue** ()
- static [DiscountFactor](#) **initialGuess** ()
- static [DiscountFactor](#) **guess** (const [YieldTermStructure](#) \*c, const [Date](#) &d)
- static [DiscountFactor](#) **minValueAfter** (Size, const std::vector< [Real](#) > &)
- static [DiscountFactor](#) **maxValueAfter** (Size i, const std::vector< [Real](#) > &data)
- static void **updateGuess** (std::vector< [DiscountFactor](#) > &data, [DiscountFactor](#) discount, Size i)

## 7.188 DiscrepancyStatistics Class Reference

```
#include <ql/Math/discrepancystatistics.hpp>
```

Inheritance diagram for DiscrepancyStatistics:



### 7.188.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

It inherit from SequenceStatistics<Statistics> and adds  $L^2$  discrepancy calculation

#### Public Types

- typedef SequenceStatistics::value\_type **value\_type**

#### Public Member Functions

- **DiscrepancyStatistics** (Size dimension)
- template<class Sequence> void **add** (const Sequence &sample, Real weight=1.0)
- template<class Iterator> void **add** (Iterator begin, Iterator end, Real weight=1.0)
- void **reset** (Size dimension=0)

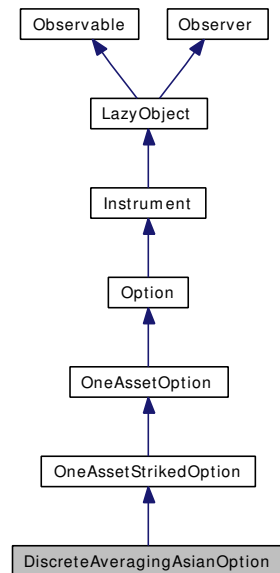
#### 1-dimensional inspectors

- Real **discrepancy** () const

## 7.189 DiscreteAveragingAsianOption Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

Inheritance diagram for DiscreteAveragingAsianOption:



### 7.189.1 Detailed Description

Discrete-averaging Asian option.

#### Public Member Functions

- **DiscreteAveragingAsianOption** (Average::Type averageType, Real runningAccumulator, Size pastFixings, std::vector< [Date](#) > fixingDates, const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) \*) const

#### Protected Attributes

- Average::Type [averageType\\_](#)
- Real [runningAccumulator\\_](#)
- Size [pastFixings\\_](#)
- std::vector< [Date](#) > [fixingDates\\_](#)

#### Classes

- class [arguments](#)

*Extra arguments for single-asset discrete-average Asian option.*



- class [engine](#)

*Discrete-averaging Asian engine base class.*

## 7.189.2 Member Function Documentation

### 7.189.2.1 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 [OneAssetStrikedOption](#).

## 7.190 DiscreteAveragingAsianOption::arguments Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

### 7.190.1 Detailed Description

Extra arguments for single-asset discrete-average Asian option.

#### Public Member Functions

- void **validate** () const

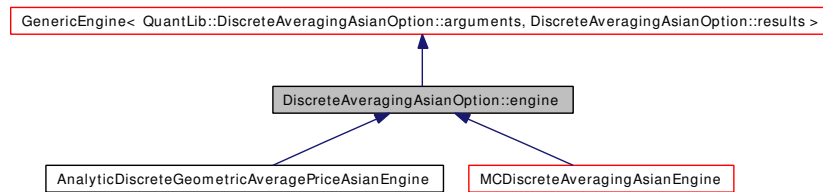
#### Public Attributes

- Average::Type **averageType**
- Real **runningAccumulator**
- Size **pastFixings**
- std::vector< [Date](#) > **fixingDates**

## 7.191 DiscreteAveragingAsianOption::engine Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

Inheritance diagram for DiscreteAveragingAsianOption::engine:



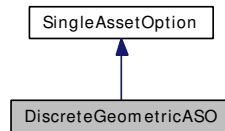
### 7.191.1 Detailed Description

Discrete-averaging Asian engine base class.

## 7.192 DiscreteGeometricASO Class Reference

```
#include <ql/Pricers/discretegeometricaso.hpp>
```

Inheritance diagram for DiscreteGeometricASO:



### 7.192.1 Detailed Description

Discrete geometric average-strike Asian option (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

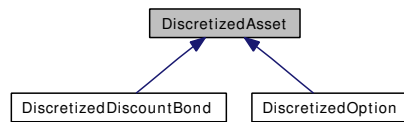
### Public Member Functions

- **DiscreteGeometricASO** (Option::Type type, [Real](#) underlying, [Spread](#) dividendYield, [Rate](#) riskFreeRate, const std::vector< Time > &times, [Volatility](#) volatility)
- [Real](#) **value** () const
- [Real](#) **delta** () const
- [Real](#) **gamma** () const
- [Real](#) **theta** () const
- boost::shared\_ptr< [SingleAssetOption](#) > **clone** () const

## 7.193 DiscretizedAsset Class Reference

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

Inheritance diagram for DiscretizedAsset:



### 7.193.1 Detailed Description

Discretized asset class used by numerical methods.

#### Public Member Functions

##### inspectors

- Time **time** () const
- Time & **time** ()
- const **Array** & **values** () const
- **Array** & **values** ()
- const boost::shared\_ptr< **NumericalMethod** > & **method** () const

##### High-level interface

Users of discretized assets should use these methods in order to initialize, evolve and take the present value of the assets. They call the corresponding methods in the NumericalMethod interface, to which we refer for documentation.

- void **initialize** (const boost::shared\_ptr< **NumericalMethod** > &, Time t)
- void **rollback** (Time to)
- void **partialRollback** (Time to)
- **Real** **presentValue** ()

##### Low-level interface

These methods (that developers should override when deriving from DiscretizedAsset) are to be used by numerical methods and not directly by users, with the exception of *adjustValues()*, *preAdjustValues()* and *postAdjustValues()* that can be used together with *partialRollback()*.

- virtual void **reset** (Size size)=0
- void **preAdjustValues** ()
- void **postAdjustValues** ()
- void **adjustValues** ()
- virtual std::vector< Time > **mandatoryTimes** () const=0

#### Protected Member Functions

- bool **isOnTime** (Time t) const
- virtual void **preAdjustValuesImpl** ()
- virtual void **postAdjustValuesImpl** ()

## Protected Attributes

- Time `time_`
- Time `latestPreAdjustment_`
- Time `latestPostAdjustment_`
- [Array](#) `values_`

## 7.193.2 Member Function Documentation

### 7.193.2.1 `virtual void reset (Size size)` [pure virtual]

This method should initialize the asset values to an [Array](#) of the given size and with values depending on the particular asset.

Implemented in [DiscretizedDiscountBond](#), and [DiscretizedOption](#).

### 7.193.2.2 `void preAdjustValues ()`

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.

This method is not virtual; derived classes must override the protected [preAdjustValuesImpl\(\)](#) method instead.

### 7.193.2.3 `void postAdjustValues ()`

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.

This method is not virtual; derived classes must override the protected [postAdjustValuesImpl\(\)](#) method instead.

### 7.193.2.4 `void adjustValues ()`

This method performs both pre- and post-adjustment

### 7.193.2.5 `virtual std::vector<Time> mandatoryTimes () const` [pure virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

#### Note:

The returned values are not guaranteed to be sorted.

Implemented in [DiscretizedDiscountBond](#), and [DiscretizedOption](#).

**7.193.2.6 bool isOnTime (Time  $t$ ) const** [protected]

This method checks whether the asset was rolled at the given time.

**7.193.2.7 virtual void preAdjustValuesImpl ()** [protected, virtual]

This method performs the actual pre-adjustment

**7.193.2.8 virtual void postAdjustValuesImpl ()** [protected, virtual]

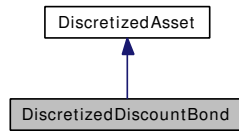
This method performs the actual post-adjustment

Reimplemented in [DiscretizedOption](#).

## 7.194 DiscretizedDiscountBond Class Reference

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

Inheritance diagram for DiscretizedDiscountBond:



### 7.194.1 Detailed Description

Useful discretized discount bond asset.

#### Public Member Functions

- void [reset](#) (Size size)
- std::vector< [Time](#) > [mandatoryTimes](#) () const

### 7.194.2 Member Function Documentation

#### 7.194.2.1 void reset (Size size) [virtual]

This method should initialize the asset values to an [Array](#) of the given size and with values depending on the particular asset.

Implements [DiscretizedAsset](#).

#### 7.194.2.2 std::vector<[Time](#)> mandatoryTimes () const [virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

#### Note:

The returned values are not guaranteed to be sorted.

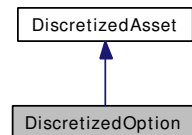
Implements [DiscretizedAsset](#).



## 7.195 DiscretizedOption Class Reference

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

Inheritance diagram for DiscretizedOption:



### 7.195.1 Detailed Description

Discretized option on a given asset.

#### Warning

it is advised that derived classes take care of creating and initializing themselves an instance of the underlying.

### Public Member Functions

- **DiscretizedOption** (const boost::shared\_ptr< [DiscretizedAsset](#) > &underlying, Exercise::Type exerciseType, const std::vector< Time > &exerciseTimes)
- void [reset](#) (Size size)
- std::vector< Time > [mandatoryTimes](#) () const

### Protected Member Functions

- void [postAdjustValuesImpl](#) ()
- void [applyExerciseCondition](#) ()

### Protected Attributes

- boost::shared\_ptr< [DiscretizedAsset](#) > **underlying\_**
- Exercise::Type **exerciseType\_**
- std::vector< Time > **exerciseTimes\_**

### 7.195.2 Member Function Documentation

#### 7.195.2.1 void reset (Size size) [virtual]

This method should initialize the asset values to an [Array](#) of the given size and with values depending on the particular asset.

Implements [DiscretizedAsset](#).

**7.195.2.2** `std::vector< Time > mandatoryTimes () const` [virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

**Note:**

The returned values are not guaranteed to be sorted.

Implements [DiscretizedAsset](#).

**7.195.2.3** `void postAdjustValuesImpl ()` [protected, virtual]

This method performs the actual post-adjustment

Reimplemented from [DiscretizedAsset](#).

## 7.196 Disposable Class Template Reference

```
#include <ql/Utilities/disposable.hpp>
```

### 7.196.1 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(Integer 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<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;  
}
```

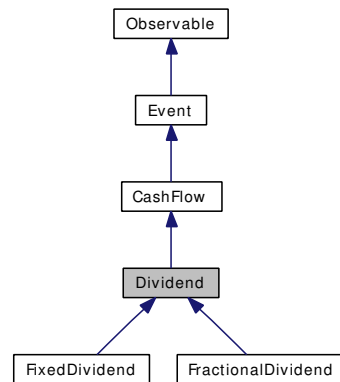
### Public Member Functions

- **Disposable** (`T &t`)
- **Disposable** (`const Disposable< T > &t`)
- `Disposable< T > &operator=` (`const Disposable< T > &t`)

## 7.197 Dividend Class Reference

```
#include <ql/CashFlows/dividend.hpp>
```

Inheritance diagram for Dividend:



### 7.197.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

### Public Member Functions

- **Dividend** (const [Date](#) &date)
- virtual Real **amount** (Real underlying) const=0

#### CashFlow interface

- virtual [Date](#) **date** () const  
*Note:*  
*This is inheirited from the event class*
- virtual Real [amount](#) () const=0  
*returns the amount of the cash flow*

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

### Protected Attributes

- [Date](#) **date\_**

## 7.197.2 Member Function Documentation

### 7.197.2.1 `virtual Real 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.

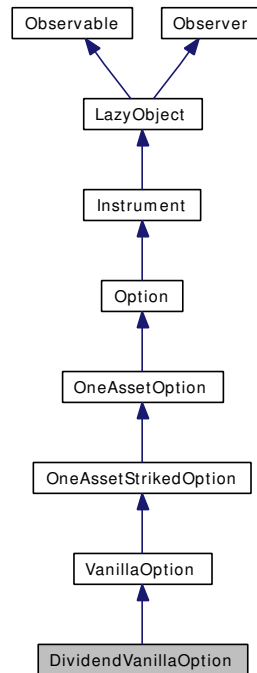
Implements [CashFlow](#).

Implemented in [FixedDividend](#), and [FractionalDividend](#).

## 7.198 DividendVanillaOption Class Reference

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

Inheritance diagram for DividendVanillaOption:



### 7.198.1 Detailed Description

Single-asset vanilla option (no barriers) with discrete dividends.

#### Public Member Functions

- **DividendVanillaOption** (const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared\_ptr< [Exercise](#) > &exercise, const std::vector< [Date](#) > &dividendDates, const std::vector< [Real](#) > &dividends, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())

#### Protected Member Functions

- void [setupArguments](#) ([Arguments](#) \*) const

#### Classes

- class [arguments](#)  
*Arguments for dividend vanilla option calculation*
- class [engine](#)

*Dividend* vanilla option engine base class.

## 7.198.2 Member Function Documentation

### 7.198.2.1 void setupArguments ([Arguments](#) \*) const [protected, 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 [OneAssetStrikedOption](#).

## 7.199 DividendVanillaOption::arguments Class Reference

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

### 7.199.1 Detailed Description

Arguments for dividend vanilla option calculation

#### Public Member Functions

- void **validate** () const

#### Public Attributes

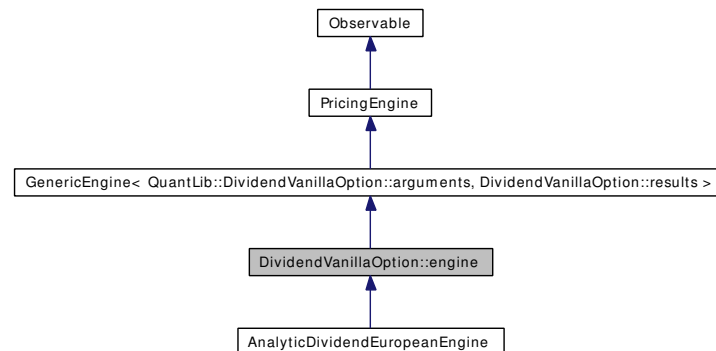
- DividendSchedule **cashFlow**



## 7.200 DividendVanillaOption::engine Class Reference

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

Inheritance diagram for DividendVanillaOption::engine:



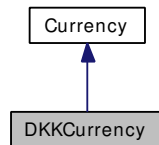
### 7.200.1 Detailed Description

[Dividend](#) vanilla option engine base class.

## 7.201 DKKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for DKKCurrency:



### 7.201.1 Detailed Description

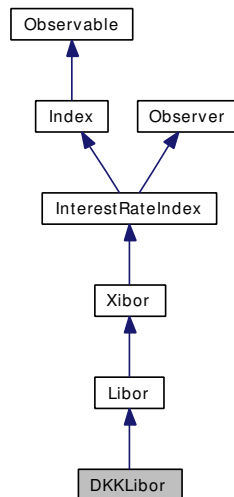
Danish krone.

The ISO three-letter code is DKK; the numeric code is 208. It is divided in 100 øre.

## 7.202 DKKLibor Class Reference

```
#include <ql/Indexes/dkklibor.hpp>
```

Inheritance diagram for DKKLibor:



### 7.202.1 Detailed Description

DKK LIBOR rate

Danish Krona LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

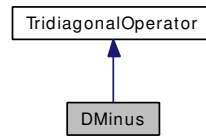
### Public Member Functions

- **DKKLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference, [Integer](#) settlementDays=2)

## 7.203 DMinus Class Reference

```
#include <ql/FiniteDifferences/dminus.hpp>
```

Inheritance diagram for DMinus:



### 7.203.1 Detailed Description

$D_-$  matricial representation

The differential operator  $D_-$  discretizes the first derivative with the first-order formula

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

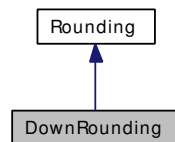
### Public Member Functions

- **DMinus** (Size gridPoints, Real h)

## 7.204 DownRounding Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for DownRounding:



### 7.204.1 Detailed Description

Down-rounding.

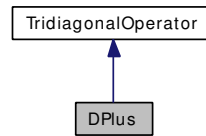
#### Public Member Functions

- **DownRounding** ([Integer](#) precision, [Integer](#) digit=5)

## 7.205 DPlus Class Reference

```
#include <ql/FiniteDifferences/dplus.hpp>
```

Inheritance diagram for DPlus:



### 7.205.1 Detailed Description

$D_+$  matricial representation

The differential operator  $D_+$  discretizes the first derivative with the first-order formula

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

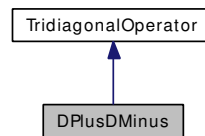
### Public Member Functions

- **DPlus** (Size gridPoints, Real h)

## 7.206 DPlusDMinus Class Reference

```
#include <ql/FiniteDifferences/dplusdminus.hpp>
```

Inheritance diagram for DPlusDMinus:



### 7.206.1 Detailed Description

$D_+D_-$  matricial representation

The differential operator  $D_+D_-$  discretizes the second derivative with the second-order formula

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

#### Tests

the correctness of the returned values is tested by checking them against numerical calculations.

### Public Member Functions

- **DPlusDMinus** (Size gridPoints, Real h)

## 7.207 DriftCalculator Class Reference

```
#include <ql/MarketModels/driftcalculator.hpp>
```

### 7.207.1 Detailed Description

Drift computation for Market Models.

### Public Member Functions

- **DriftCalculator** (const [Matrix](#) &pseudo, const std::vector< [Rate](#) > &displacements, const std::vector< [Time](#) > &taus, Size numeraire, Size alive)
- void **compute** (const std::vector< [Rate](#) > &forwards, std::vector< [Real](#) > &drifts) const  
*Computes the drifts.*
- void **computePlain** (const std::vector< [Rate](#) > &forwards, std::vector< [Real](#) > &drifts) const
- void **computeReduced** (const std::vector< [Rate](#) > &forwards, std::vector< [Real](#) > &drifts) const

### 7.207.2 Constructor & Destructor Documentation

**7.207.2.1 [DriftCalculator](#)** (const [Matrix](#) & pseudo, const std::vector< [Rate](#) > & displacements, const std::vector< [Time](#) > & taus, Size numeraire, Size alive)

Returns the drift  $\mu\Delta t$ .

See [1] "Rapid Computation of Drifts in a Reduced Factor [Libor](#) Market Model" Mark Joshi, Wilmott Magazine, May 2003.

### 7.207.3 Member Function Documentation

**7.207.3.1 void computePlain** (const std::vector< [Rate](#) > & forwards, std::vector< [Real](#) > & drifts) const

Computes the drifts without factor reduction as in eqs. 2, 4 of ref. [1] (uses the covariance matrix directly).

**7.207.3.2 void computeReduced** (const std::vector< [Rate](#) > & forwards, std::vector< [Real](#) > & drifts) const

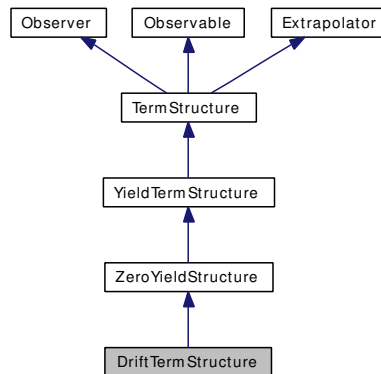
Computes the drifts with factor reduction as in eq. 7 of ref. [1] (uses pseudo square root of the covariance matrix).



## 7.208 DriftTermStructure Class Reference

```
#include <ql/TermStructures/drifttermstructure.hpp>
```

Inheritance diagram for DriftTermStructure:



### 7.208.1 Detailed Description

Drift term structure.

Drift term structure for modelling the common drift term:  $\text{riskFreeRate} - \text{dividendYield} - 0.5 \cdot \text{vol} \cdot \text{vol}$

#### 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 [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > &dividendTS, const [Handle](#)< [BlackVolTermStructure](#) > &blackVolTS)

#### YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Calendar](#) [calendar](#) () const  
*the calendar used for reference date calculation*
- const [Date](#) & [referenceDate](#) () const  
*the date at which discount = 1.0 and/or variance = 0.0*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return values*

## Protected Member Functions

- [Rate zeroYieldImpl](#) (Time) const  
*returns the discount factor as seen from the evaluation date*

## 7.209 Duration Struct Reference

```
#include <ql/CashFlows/analysis.hpp>
```

### 7.209.1 Detailed Description

duration type

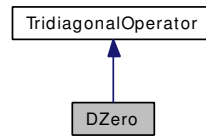
#### Public Types

- enum Type { Simple, Macaulay, Modified }

## 7.210 DZero Class Reference

```
#include <ql/FiniteDifferences/dzero.hpp>
```

Inheritance diagram for DZero:



### 7.210.1 Detailed Description

$D_0$  matricial representation

The differential operator  $D_0$  discretizes the first derivative with the second-order formula

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

#### Tests

the correctness of the returned values is tested by checking them against numerical calculations.

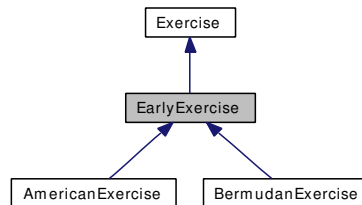
### Public Member Functions

- **DZero** (Size gridPoints, Real h)

## 7.211 EarlyExercise Class Reference

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

Inheritance diagram for EarlyExercise:



### 7.211.1 Detailed Description

Early-exercise base class.

The payoff can be at exercise (the default) or at expiry

#### Public Member Functions

- **EarlyExercise** (Type type, bool payoffAtExpiry=false)
- bool **payoffAtExpiry** () const

## 7.212 EarlyExercisePathPricer Class Template Reference

```
#include <ql/MonteCarlo/earlyexercisepathpricer.hpp>
```

### 7.212.1 Detailed Description

**template<class PathType, class TimeType = Size, class ValueType = Real> class QuantLib::EarlyExercisePathPricer< PathType, TimeType, ValueType >**

base class for early exercise path pricers

Returns the value of an option on a given path and given time.

### Public Types

- **typedef** EarlyExerciseTraits< PathType >::StateType **StateType**

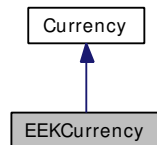
### Public Member Functions

- virtual ValueType **operator()** (const PathType &path, TimeType t) const=0
- virtual StateType **state** (const PathType &path, TimeType t) const=0
- virtual std::vector< boost::function1< ValueType, StateType > > **basisSystem** () const=0

## 7.213 EEKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for EEKCurrency:



### 7.213.1 Detailed Description

Estonian kroon.

The ISO three-letter code is EEK; the numeric code is 233. It is divided in 100 senti.

## 7.214 EndCriteria Class Reference

```
#include <ql/Optimization/criteria.hpp>
```

### 7.214.1 Detailed Description

Criteria to end optimization process.

- stationary point
  - stationary gradient
  - maximum number of iterations ....

### Public Types

- enum **Type** { **none**, **maxIter**, **statPt**, **statGd** }

### Public Member Functions

- [EndCriteria](#) ()  
*default constructor*
- [EndCriteria](#) (Size maxIteration, Real epsilon)  
*initialization constructor*
- void **setPositiveOptimization** (bool)
- bool [operator\(\)](#) (Size iteration, Real fold, Real normgold, Real fnew, Real normgnew, Real)
- Type [criteria](#) () const  
*return the end criteria type*
- bool **checkIterationNumber** (Size iteration)
- bool **checkStationaryValue** (Real fold, Real fnew)
- bool **checkAccuracyValue** (Real f)
- bool **checkStationaryGradientNorm** (Real normDiff)
- bool **checkAccuracyGradientNorm** (Real norm)

### Protected Attributes

- Size [maxIteration\\_](#)  
*Maximum number of iterations.*
- Real [functionEpsilon\\_](#)  
*function and gradient epsilons*
- Real [gradientEpsilon\\_](#)
- Size [maxIterStatPt\\_](#)  
*Maximun number of iterations in stationary state.*



- Size `statState_`
- Type `endCriteria_`
- bool `positiveOptimization_`

## 7.214.2 Member Function Documentation

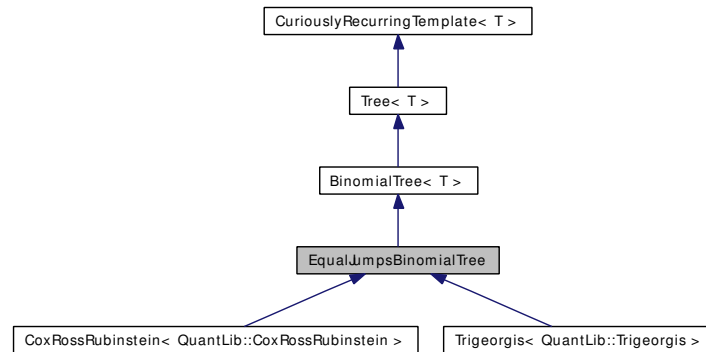
### 7.214.2.1 `bool operator()` (Size *iteration*, Real *fold*, Real *normgold*, Real *fnew*, Real *normgnew*, Real)

test if the number of iteration is not too big and if we don't raise a stationary point

## 7.215 EqualJumpsBinomialTree Class Template Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for EqualJumpsBinomialTree:



### 7.215.1 Detailed Description

```
template<class T> class QuantLib::EqualJumpsBinomialTree< T >
```

Base class for equal jumps binomial tree.

#### Public Member Functions

- **EqualJumpsBinomialTree** (const boost::shared\_ptr< [StochasticProcess1D](#) > &process, [Time](#) end, Size steps)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

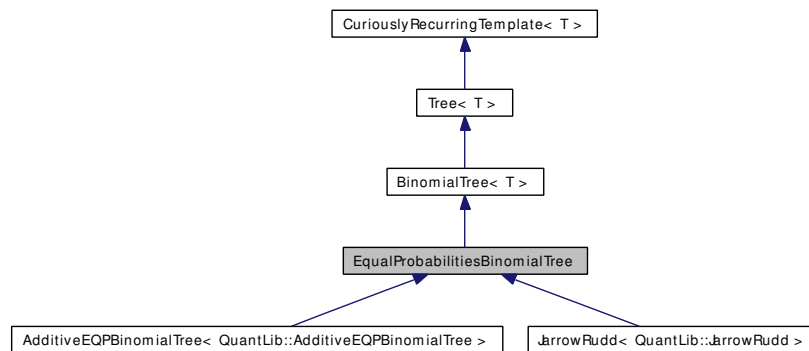
#### Protected Attributes

- Real **dx\_**
- Real **pu\_**
- Real **pd\_**

## 7.216 EqualProbabilitiesBinomialTree Class Template Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for EqualProbabilitiesBinomialTree:



### 7.216.1 Detailed Description

```
template<class T> class QuantLib::EqualProbabilitiesBinomialTree< T >
```

Base class for equal probabilities binomial tree.

#### Public Member Functions

- **EqualProbabilitiesBinomialTree** (const boost::shared\_ptr< [StochasticProcess1D](#) > &process, [Time](#) end, Size steps)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size) const

#### Protected Attributes

- Real **up\_**

## 7.217 Error Class Reference

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

### 7.217.1 Detailed Description

Base error class.

### Public Member Functions

- [Error](#) (const std::string &file, long line, const std::string &function, const std::string &message="")
- [~Error](#) () throw ()
- const char \* [what](#) () const throw ()  
*returns the error message.*

### 7.217.2 Constructor & Destructor Documentation

**7.217.2.1 [Error](#) (const std::string &file, long line, const std::string &function, const std::string &message = "")**

The explicit use of this constructor is not advised. Use the QL\_FAIL macro instead.

**7.217.2.2 [~Error](#) () throw ()**

the automatically generated destructor would not have the throw specifier.

## 7.218 ErrorFunction Class Reference

```
#include <ql/Math/errorfunction.hpp>
```

### 7.218.1 Detailed Description

Error function

formula here ... Used to calculate the cumulative normal distribution function

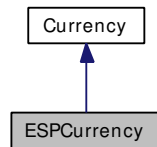
### Public Member Functions

- Real **operator()** (Real x) const

## 7.219 ESPCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ESPCurrency:



### 7.219.1 Detailed Description

Spanish peseta.

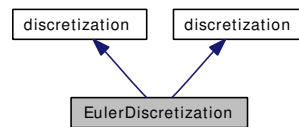
The ISO three-letter code was ESP; the numeric code was 724. It was divided in 100 centimos.

Obsoleted by the Euro since 1999.

## 7.220 EulerDiscretization Class Reference

```
#include <ql/Processes/eulerdiscretization.hpp>
```

Inheritance diagram for EulerDiscretization:



### 7.220.1 Detailed Description

Euler discretization for stochastic processes.

#### Public Member Functions

- `Disposable< Array > drift` (const `StochasticProcess` &, Time `t0`, const `Array` &`x0`, Time `dt`) const
- `Real drift` (const `StochasticProcess1D` &, Time `t0`, Real `x0`, Time `dt`) const
- `Disposable< Matrix > diffusion` (const `StochasticProcess` &, Time `t0`, const `Array` &`x0`, Time `dt`) const
- `Real diffusion` (const `StochasticProcess1D` &, Time `t0`, Real `x0`, Time `dt`) const
- `Disposable< Matrix > covariance` (const `StochasticProcess` &, Time `t0`, const `Array` &`x0`, Time `dt`) const
- `Real variance` (const `StochasticProcess1D` &, Time `t0`, Real `x0`, Time `dt`) const

### 7.220.2 Member Function Documentation

**7.220.2.1** `Disposable<Array> drift` (const `StochasticProcess` &, Time `t0`, const `Array` & `x0`, Time `dt`) const [virtual]

Returns an approximation of the drift defined as  $\mu(t_0, x_0)\Delta t$ .

Implements `StochasticProcess::discretization`.

**7.220.2.2** `Real drift` (const `StochasticProcess1D` &, Time `t0`, Real `x0`, Time `dt`) const [virtual]

Returns an approximation of the drift defined as  $\mu(t_0, x_0)\Delta t$ .

Implements `StochasticProcess1D::discretization`.

**7.220.2.3** `Disposable<Matrix> diffusion` (const `StochasticProcess` &, Time `t0`, const `Array` & `x0`, Time `dt`) const [virtual]

Returns an approximation of the diffusion defined as  $\sigma(t_0, x_0) \sqrt{\Delta t}$ .

Implements `StochasticProcess::discretization`.

**7.220.2.4 Real diffusion** (const [StochasticProcess1D](#) &, Time *t0*, Real *x0*, Time *dt*) const  
[virtual]

Returns an approximation of the diffusion defined as  $\sigma(t_0, x_0) \sqrt{\Delta t}$ .

Implements [StochasticProcess1D::discretization](#).

**7.220.2.5 Disposable<Matrix> covariance** (const [StochasticProcess](#) &, Time *t0*, const [Array](#) & *x0*, Time *dt*) const [virtual]

Returns an approximation of the covariance defined as  $\sigma(t_0, x_0)^2 \Delta t$ .

Implements [StochasticProcess::discretization](#).

**7.220.2.6 Real variance** (const [StochasticProcess1D](#) &, Time *t0*, Real *x0*, Time *dt*) const  
[virtual]

Returns an approximation of the variance defined as  $\sigma(t_0, x_0)^2 \Delta t$ .

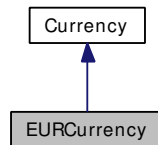
Implements [StochasticProcess1D::discretization](#).



## 7.221 EURCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for EURCurrency:



### 7.221.1 Detailed Description

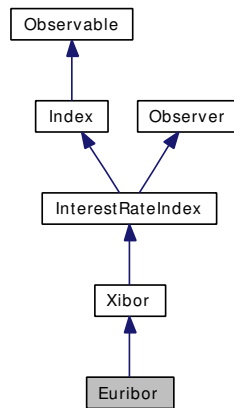
European Euro.

The ISO three-letter code is EUR; the numeric code is 978. It is divided into 100 cents.

## 7.222 Euribor Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

Inheritance diagram for Euribor:



### 7.222.1 Detailed Description

Euribor index

**Euribor** rate fixed by the ECB.

#### Warning

This is the rate fixed by the ECB. Use **EurLibor** if you're interested in the London fixing by BBA.

### Public Member Functions

- **Euribor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference)

## 7.223 Euribor10M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.223.1 Detailed Description

10-months Euribor index

#### Public Member Functions

- **Euribor10M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.224 Euribor11M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.224.1 Detailed Description

11-months Euribor index

#### Public Member Functions

- **Euribor11M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.225 Euribor1M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.225.1 Detailed Description

1-month Euribor index

#### Public Member Functions

- **Euribor1M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.226 Euribor1Y Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.226.1 Detailed Description

1-year Euribor index

#### Public Member Functions

- **Euribor1Y** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.227 Euribor2M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.227.1 Detailed Description

2-months Euribor index

#### Public Member Functions

- **Euribor2M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.228 Euribor2W Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.228.1 Detailed Description

2-weeks Euribor index

### Public Member Functions

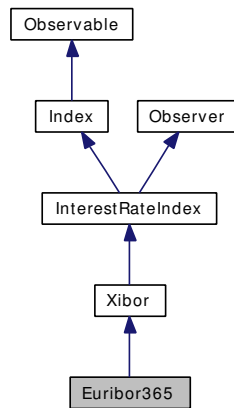
- **Euribor2W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())



## 7.229 Euribor365 Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

Inheritance diagram for Euribor365:



### 7.229.1 Detailed Description

Actual/365 Euribor index.

**Euribor** rate adjusted for the mismatch between the actual/360 convention used for **Euribor** and the actual/365 convention previously used by a few pre-EUR currencies.

### Public Member Functions

- **Euribor365** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference)

## 7.230 Euribor365\_10M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.230.1 Detailed Description

10-months Euribor365 index

#### Public Member Functions

- **Euribor365\_10M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.231 Euribor365\_11M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.231.1 Detailed Description

11-months Euribor365 index

#### Public Member Functions

- **Euribor365\_11M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.232 Euribor365\_1M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.232.1 Detailed Description

1-month Euribor365 index

#### Public Member Functions

- **Euribor365\_1M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.233 Euribor365\_1Y Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.233.1 Detailed Description

1-year Euribor365 index

#### Public Member Functions

- `Euribor365_1Y(const Handle<YieldTermStructure> &h=Handle<YieldTermStructure>())`

## 7.234 Euribor365\_2M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.234.1 Detailed Description

2-months Euribor365 index

#### Public Member Functions

- **Euribor365\_2M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.235 Euribor365\_2W Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.235.1 Detailed Description

2-weeks Euribor365 index

#### Public Member Functions

- **Euribor365\_2W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.236 Euribor365\_3M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.236.1 Detailed Description

3-months Euribor365 index

#### Public Member Functions

- **Euribor365\_3M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())



## 7.237 Euribor365\_3W Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.237.1 Detailed Description

3-weeks Euribor365 index

#### Public Member Functions

- **Euribor365\_3W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.238 Euribor365\_4M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.238.1 Detailed Description

4-months Euribor365 index

### Public Member Functions

- **Euribor365\_4M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.239 Euribor365\_5M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.239.1 Detailed Description

5-months Euribor365 index

#### Public Member Functions

- **Euribor365\_5M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.240 Euribor365\_6M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.240.1 Detailed Description

6-months Euribor365 index

### Public Member Functions

- **Euribor365\_6M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.241 Euribor365\_7M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.241.1 Detailed Description

7-months Euribor365 index

#### Public Member Functions

- **Euribor365\_7M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.242 Euribor365\_8M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.242.1 Detailed Description

8-months Euribor365 index

### Public Member Functions

- **Euribor365\_8M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.243 Euribor365\_9M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.243.1 Detailed Description

9-months Euribor365 index

#### Public Member Functions

- **Euribor365\_9M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.244 Euribor365\_SW Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.244.1 Detailed Description

1-week Euribor365 index

#### Public Member Functions

- **Euribor365\_SW** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())



## 7.245 Euribor3M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.245.1 Detailed Description

3-months Euribor index

Examples:

[FRA.cpp](#).

### Public Member Functions

- **Euribor3M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.246 Euribor3W Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.246.1 Detailed Description

3-weeks Euribor index

#### Public Member Functions

- **Euribor3W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.247 Euribor4M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.247.1 Detailed Description

4-months Euribor index

#### Public Member Functions

- **Euribor4M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.248 Euribor5M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.248.1 Detailed Description

5-months Euribor index

### Public Member Functions

- **Euribor5M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.249 Euribor6M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.249.1 Detailed Description

6-months Euribor index

**Examples:**

[BermudanSwaption.cpp](#), and [swapvaluation.cpp](#).

### Public Member Functions

- **Euribor6M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.250 Euribor7M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.250.1 Detailed Description

7-months Euribor index

### Public Member Functions

- **Euribor7M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.251 Euribor8M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.251.1 Detailed Description

8-months Euribor index

### Public Member Functions

- **Euribor8M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.252 Euribor9M Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.252.1 Detailed Description

9-months Euribor index

### Public Member Functions

- **Euribor9M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())



## 7.253 EuriborSW Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

### 7.253.1 Detailed Description

1-week Euribor index

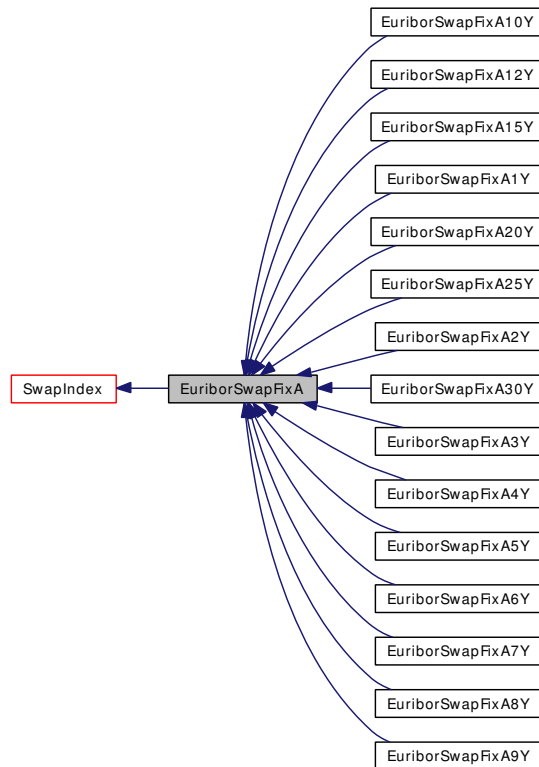
#### Public Member Functions

- **EuriborSW** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.254 EuriborSwapFixA Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA:



### 7.254.1 Detailed Description

EuriborSwapFixA index

[EuriborSwapFixA](#) rate fixed by ISDA. The swap index is based on the [Euribor](#) 6M and is fixed at 11:00AM FRANKFURT. Reuters page ISDAFIX2 or EURSFXA=.

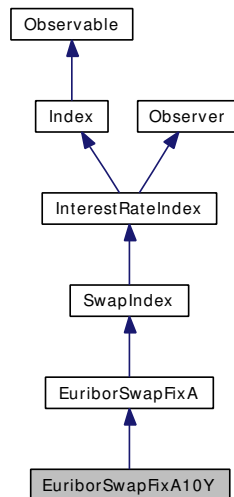
### Public Member Functions

- [EuriborSwapFixA](#) ([Integer](#) years, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.255 EuriborSwapFixA10Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA10Y:



### 7.255.1 Detailed Description

10-year EuriborSwapFixA index

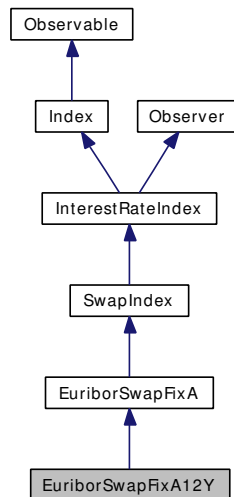
#### Public Member Functions

- `EuriborSwapFixA10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.256 EuriborSwapFixA12Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA12Y:



### 7.256.1 Detailed Description

12-year EuriborSwapFixA index

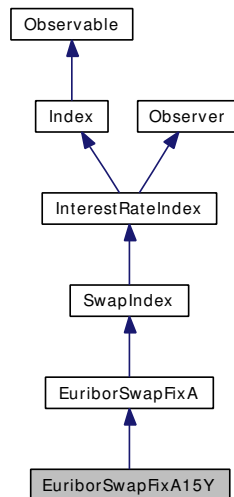
#### Public Member Functions

- `EuriborSwapFixA12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.257 EuriborSwapFixA15Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA15Y:



### 7.257.1 Detailed Description

15-year EuriborSwapFixA index

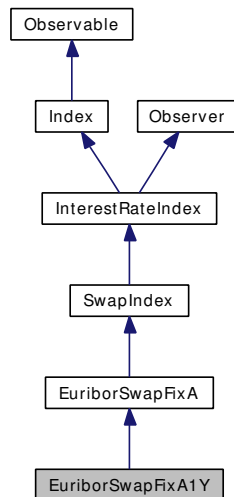
#### Public Member Functions

- EuriborSwapFixA15Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.258 EuriborSwapFixA1Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA1Y:



### 7.258.1 Detailed Description

1-year EuriborSwapFixA index

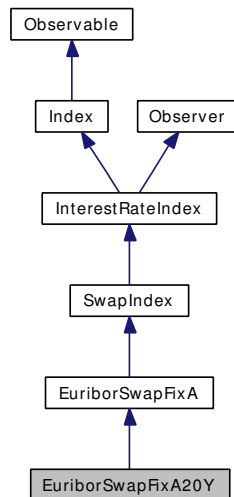
#### Public Member Functions

- `EuriborSwapFixA1Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.259 EuriborSwapFixA20Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA20Y:



### 7.259.1 Detailed Description

20-year EuriborSwapFixA index

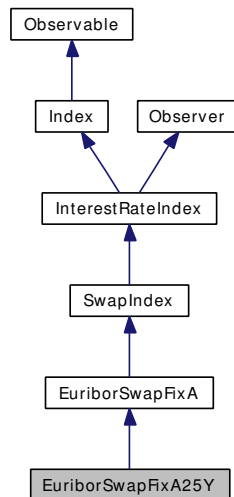
#### Public Member Functions

- `EuriborSwapFixA20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.260 EuriborSwapFixA25Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA25Y:



### 7.260.1 Detailed Description

25-year EuriborSwapFixA index

#### Public Member Functions

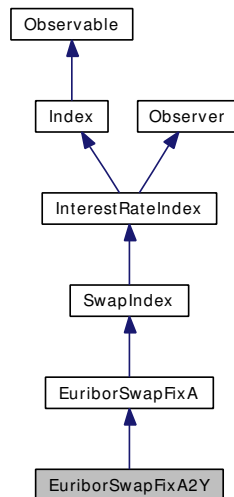
- EuriborSwapFixA25Y (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.261 EuriborSwapFixA2Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA2Y:



### 7.261.1 Detailed Description

2-year EuriborSwapFixA index

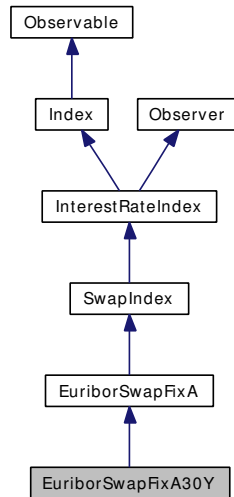
#### Public Member Functions

- `EuriborSwapFixA2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.262 EuriborSwapFixA30Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA30Y:



### 7.262.1 Detailed Description

30-year EuriborSwapFixA index

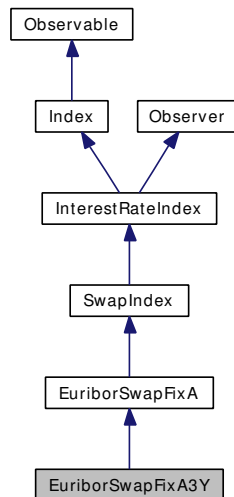
#### Public Member Functions

- `EuriborSwapFixA30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.263 EuriborSwapFixA3Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA3Y:



### 7.263.1 Detailed Description

3-year EuriborSwapFixA index

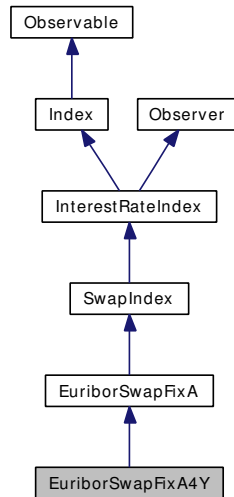
#### Public Member Functions

- `EuriborSwapFixA3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.264 EuriborSwapFixA4Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA4Y:



### 7.264.1 Detailed Description

4-year EuriborSwapFixA index

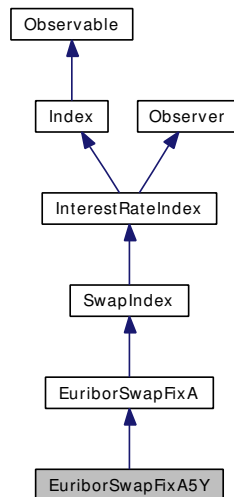
#### Public Member Functions

- `EuriborSwapFixA4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.265 EuriborSwapFixA5Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA5Y:



### 7.265.1 Detailed Description

5-year EuriborSwapFixA index

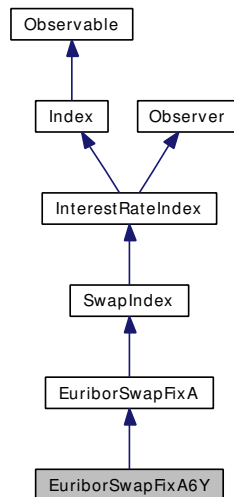
#### Public Member Functions

- EuriborSwapFixA5Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.266 EuriborSwapFixA6Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA6Y:



### 7.266.1 Detailed Description

6-year EuriborSwapFixA index

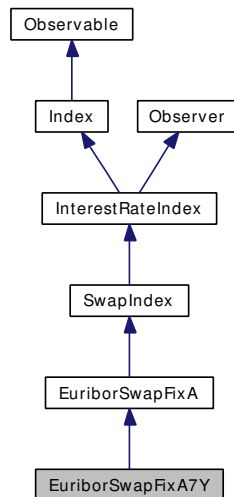
#### Public Member Functions

- EuriborSwapFixA6Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.267 EuriborSwapFixA7Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA7Y:



### 7.267.1 Detailed Description

7-year EuriborSwapFixA index

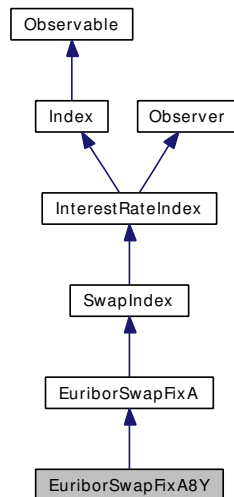
#### Public Member Functions

- EuriborSwapFixA7Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.268 EuriborSwapFixA8Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA8Y:



### 7.268.1 Detailed Description

8-year EuriborSwapFixA index

#### Public Member Functions

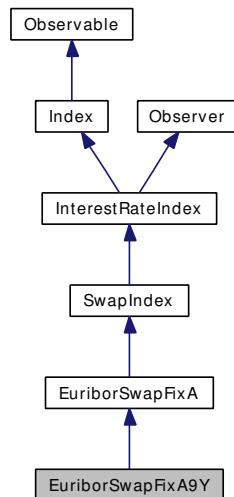
- EuriborSwapFixA8Y (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.269 EuriborSwapFixA9Y Class Reference

```
#include <ql/Indexes/euriborswapfixa.hpp>
```

Inheritance diagram for EuriborSwapFixA9Y:



### 7.269.1 Detailed Description

9-year EuriborSwapFixA index

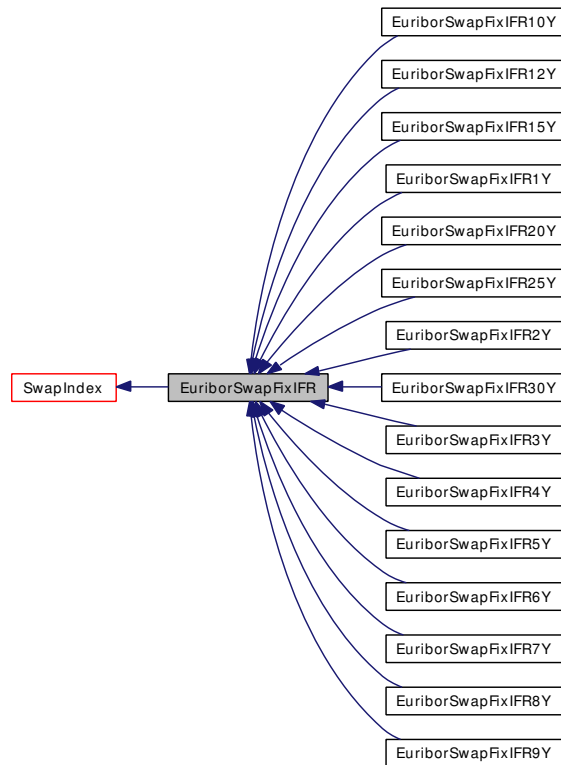
#### Public Member Functions

- `EuriborSwapFixA9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.270 EuriborSwapFixIFR Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR:



### 7.270.1 Detailed Description

EuriborSwapFixIFR index

EuriborSwapFix index published by IFR Markets and distributed by Reuters page TGM42281 and by Telerate. For more info see <<http://www.ifrmarkets.com>>.

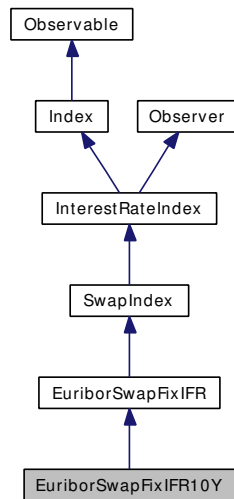
### Public Member Functions

- **EuriborSwapFixIFR** (*Integer* years, const *Handle*< *YieldTermStructure* > &h=*Handle*< *YieldTermStructure* >())

## 7.271 EuriborSwapFixIFR10Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR10Y:



### 7.271.1 Detailed Description

10-year EuriborSwapFixIFR index

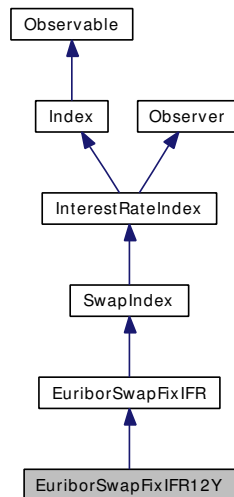
#### Public Member Functions

- `EuriborSwapFixIFR10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.272 EuriborSwapFixIFR12Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR12Y:



### 7.272.1 Detailed Description

12-year `EuriborSwapFixIFR` index

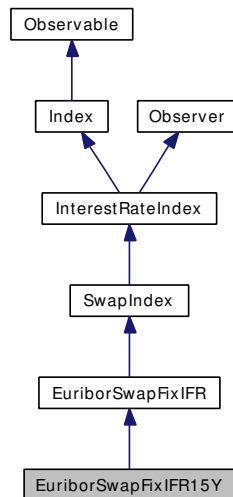
#### Public Member Functions

- `EuriborSwapFixIFR12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.273 EuriborSwapFixIFR15Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR15Y:



### 7.273.1 Detailed Description

15-year EuriborSwapFixIFR index

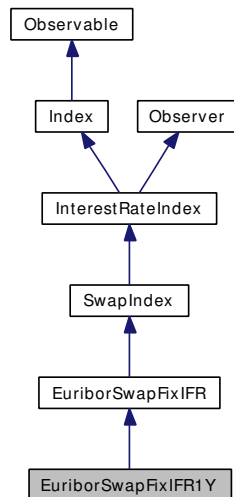
#### Public Member Functions

- `EuriborSwapFixIFR15Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.274 EuriborSwapFixIFR1Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR1Y:



### 7.274.1 Detailed Description

1-year EuriborSwapFixIFR index

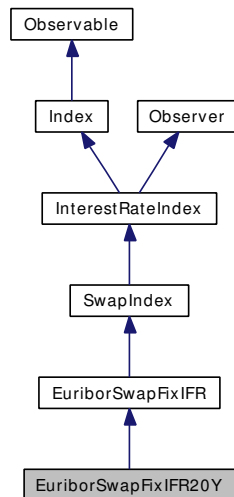
#### Public Member Functions

- EuriborSwapFixIFR1Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.275 EuriborSwapFixIFR20Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR20Y:



### 7.275.1 Detailed Description

20-year EuriborSwapFixIFR index

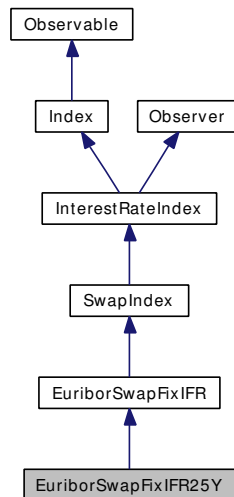
#### Public Member Functions

- `EuriborSwapFixIFR20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.276 EuriborSwapFixIFR25Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR25Y:



### 7.276.1 Detailed Description

25-year EuriborSwapFixIFR index

#### Public Member Functions

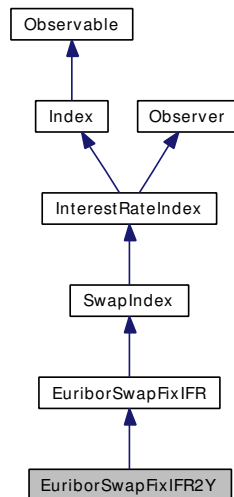
- `EuriborSwapFixIFR25Y` (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.277 EuriborSwapFixIFR2Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR2Y:



### 7.277.1 Detailed Description

2-year EuriborSwapFixIFR index

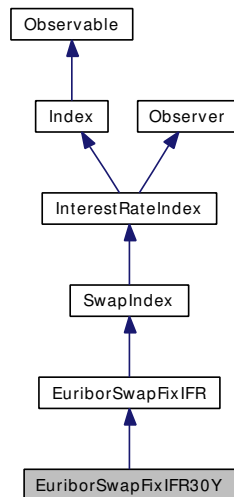
#### Public Member Functions

- `EuriborSwapFixIFR2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.278 EuriborSwapFixIFR30Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR30Y:



### 7.278.1 Detailed Description

30-year EuriborSwapFixIFR index

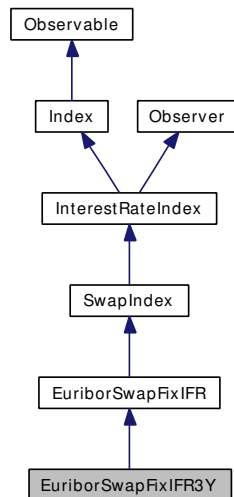
#### Public Member Functions

- `EuriborSwapFixIFR30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.279 EuriborSwapFixIFR3Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR3Y:



### 7.279.1 Detailed Description

3-year EuriborSwapFixIFR index

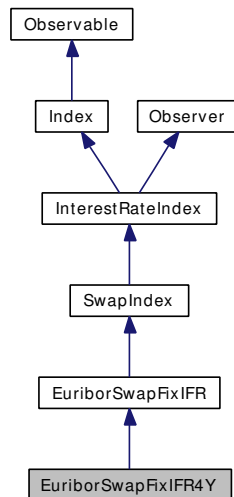
#### Public Member Functions

- `EuriborSwapFixIFR3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.280 EuriborSwapFixIFR4Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR4Y:



### 7.280.1 Detailed Description

4-year EuriborSwapFixIFR index

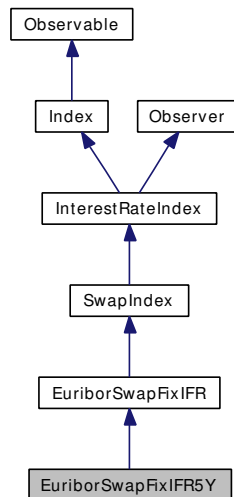
#### Public Member Functions

- `EuriborSwapFixIFR4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.281 EuriborSwapFixIFR5Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR5Y:



### 7.281.1 Detailed Description

5-year EuriborSwapFixIFR index

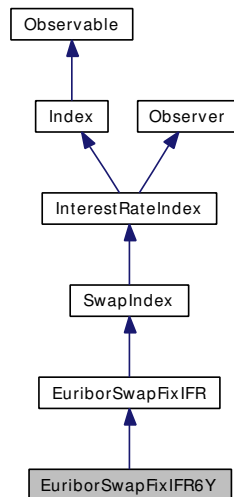
#### Public Member Functions

- `EuriborSwapFixIFR5Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.282 EuriborSwapFixIFR6Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR6Y:



### 7.282.1 Detailed Description

6-year EuriborSwapFixIFR index

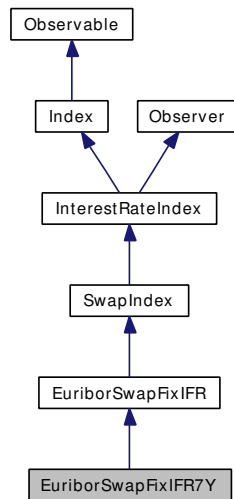
#### Public Member Functions

- `EuriborSwapFixIFR6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.283 EuriborSwapFixIFR7Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR7Y:



### 7.283.1 Detailed Description

7-year EuriborSwapFixIFR index

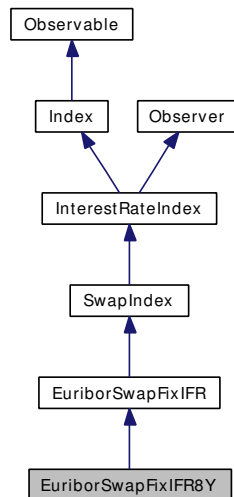
#### Public Member Functions

- `EuriborSwapFixIFR7Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.284 EuriborSwapFixIFR8Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR8Y:



### 7.284.1 Detailed Description

8-year EuriborSwapFixIFR index

#### Public Member Functions

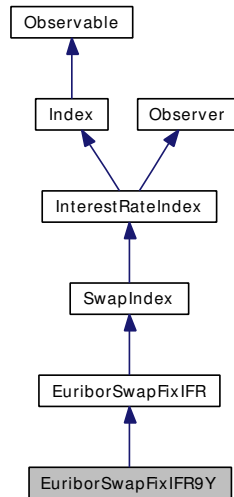
- EuriborSwapFixIFR8Y (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.285 EuriborSwapFixIFR9Y Class Reference

```
#include <ql/Indexes/euriborswapfixifr.hpp>
```

Inheritance diagram for EuriborSwapFixIFR9Y:



### 7.285.1 Detailed Description

9-year EuriborSwapFixIFR index

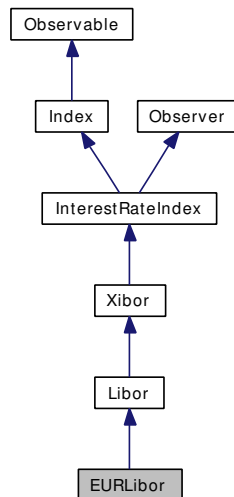
#### Public Member Functions

- `EuriborSwapFixIFR9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.286 EURLibor Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

Inheritance diagram for EURLibor:



### 7.286.1 Detailed Description

EUR LIBOR rate

Euro LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

#### Warning

This is the rate fixed in London by BBA. Use [Euribor](#) if you're interested in the fixing by the ECB.

### Public Member Functions

- **EURLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference, [Integer](#) settlementDays=2)

## 7.287 EURLibor10M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.287.1 Detailed Description

10-months EURLibor index

#### Public Member Functions

- `EURLibor10M(const Handle<YieldTermStructure> &h=Handle<YieldTermStructure>())`

## 7.288 EURLibor11M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.288.1 Detailed Description

11-months EURLibor index

#### Public Member Functions

- `EURLibor11M(const Handle< YieldTermStructure > &h=Handle< YieldTermStructure >())`

## 7.289 EURLibor1M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.289.1 Detailed Description

1-month EURLibor index

#### Public Member Functions

- **EURLibor1M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.290 EURLibor1Y Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.290.1 Detailed Description

1-year EURLibor index

### Public Member Functions

- **EURLibor1Y** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.291 EURLibor2M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.291.1 Detailed Description

2-months EURLibor index

#### Public Member Functions

- **EURLibor2M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.292 EURLibor2W Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.292.1 Detailed Description

2-weeks Euribor index

### Public Member Functions

- **EURLibor2W** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())



## 7.293 EURLibor3M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.293.1 Detailed Description

3-months EURLibor index

#### Public Member Functions

- **EURLibor3M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.294 EURLibor4M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.294.1 Detailed Description

4-months EURLibor index

#### Public Member Functions

- **EURLibor4M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.295 EURLibor5M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.295.1 Detailed Description

5-months EURLibor index

#### Public Member Functions

- **EURLibor5M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.296 EURLibor6M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.296.1 Detailed Description

6-months EURLibor index

#### Public Member Functions

- **EURLibor6M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.297 EURLibor7M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.297.1 Detailed Description

7-months EURLibor index

#### Public Member Functions

- **EURLibor7M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.298 EURLibor8M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.298.1 Detailed Description

8-months EURLibor index

### Public Member Functions

- **EURLibor8M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.299 EURLibor9M Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.299.1 Detailed Description

9-months EURLibor index

#### Public Member Functions

- **EURLibor9M** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.300 EURLiborSW Class Reference

```
#include <ql/Indexes/eurlibor.hpp>
```

### 7.300.1 Detailed Description

1-week EURLibor index

### Public Member Functions

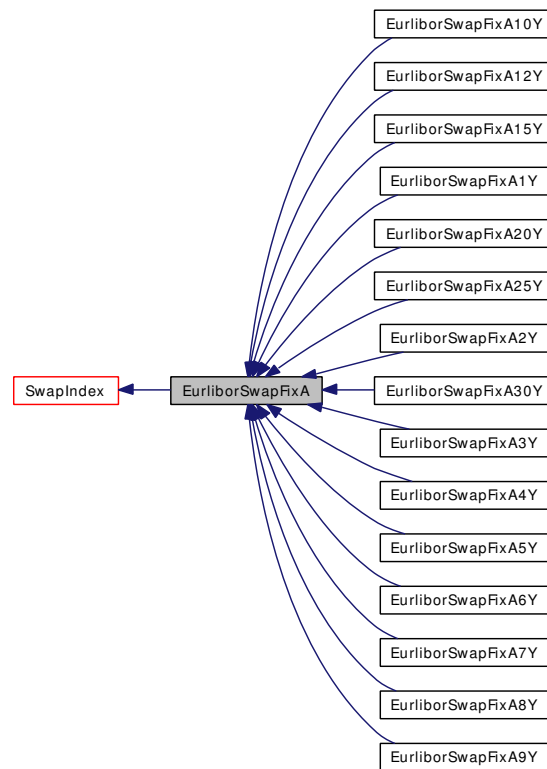
- **EURLiborSW** (const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())



## 7.301 EurliborSwapFixA Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA:



### 7.301.1 Detailed Description

EurliborSwapFixA index

[EurliborSwapFixA](#) rate fixed by ISDA in cooperation with Reuters and Intercapital Brokers. The swap index is based on the EuroLibor 6M and is fixed at 10:00 AM London. Reuters page ISDAFIX2 or EURLIBORFIXLA=. Further info can be found at: <http://www.isda.org/fix/isdafix.html>.

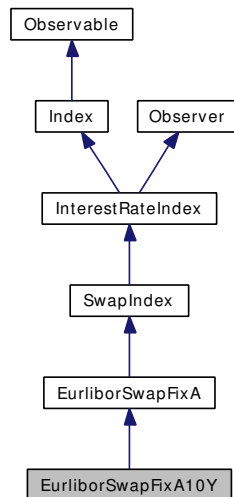
### Public Member Functions

- [EurliborSwapFixA](#) ([Integer](#) years, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.302 EurliborSwapFixA10Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA10Y:



### 7.302.1 Detailed Description

10-year EurliborSwapFixA index

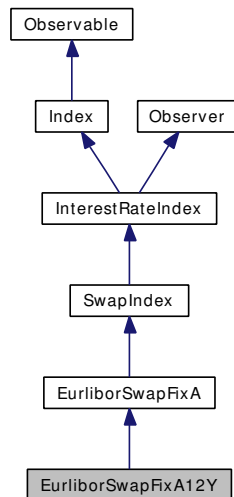
#### Public Member Functions

- `EurliborSwapFixA10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.303 EurliborSwapFixA12Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA12Y:



### 7.303.1 Detailed Description

12-year EurliborSwapFixA index

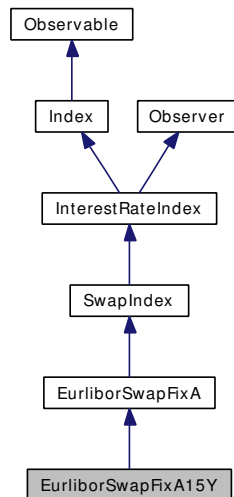
#### Public Member Functions

- `EurliborSwapFixA12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.304 EurliborSwapFixA15Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA15Y:



### 7.304.1 Detailed Description

15-year EurliborSwapFixA index

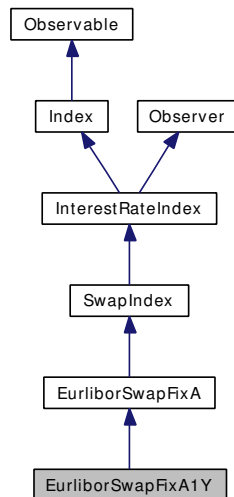
#### Public Member Functions

- `EurliborSwapFixA15Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.305 EurliborSwapFixA1Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA1Y:



### 7.305.1 Detailed Description

1-year EurliborSwapFixA index

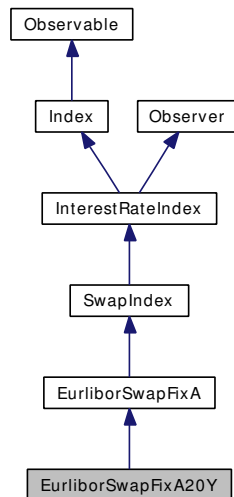
#### Public Member Functions

- `EurliborSwapFixA1Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.306 EurliborSwapFixA20Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA20Y:



### 7.306.1 Detailed Description

20-year EurliborSwapFixA index

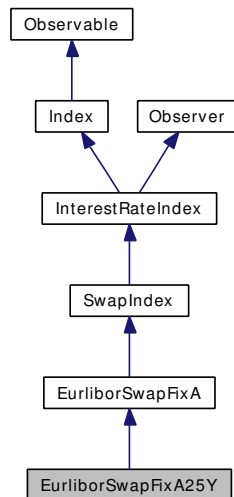
#### Public Member Functions

- `EurliborSwapFixA20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.307 EurliborSwapFixA25Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA25Y:



### 7.307.1 Detailed Description

25-year EurliborSwapFixA index

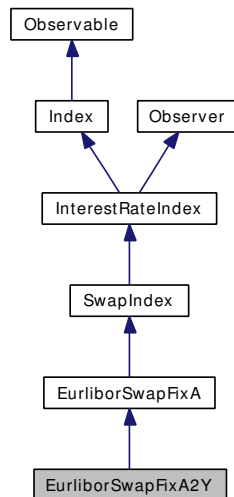
#### Public Member Functions

- `EurliborSwapFixA25Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.308 EurliborSwapFixA2Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA2Y:



### 7.308.1 Detailed Description

2-year EurliborSwapFixA index

#### Public Member Functions

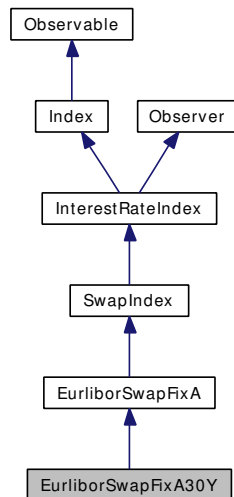
- `EurliborSwapFixA2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.309 EurliborSwapFixA30Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA30Y:



### 7.309.1 Detailed Description

30-year EurliborSwapFixA index

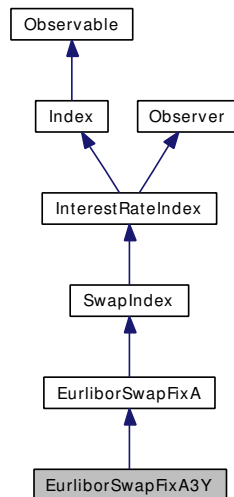
#### Public Member Functions

- `EurliborSwapFixA30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.310 EurliborSwapFixA3Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA3Y:



### 7.310.1 Detailed Description

3-year EurliborSwapFixA index

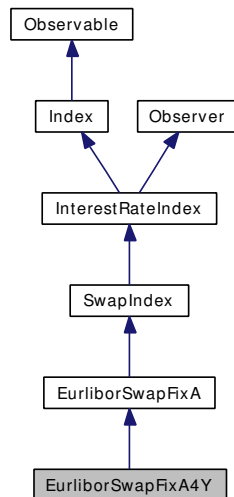
#### Public Member Functions

- EurliborSwapFixA3Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.311 EurliborSwapFixA4Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA4Y:



### 7.311.1 Detailed Description

4-year EurliborSwapFixA index

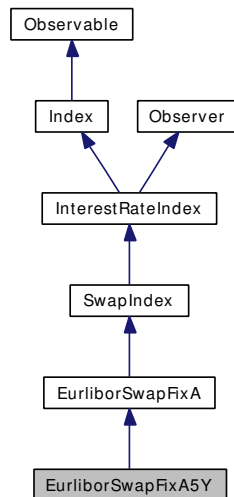
#### Public Member Functions

- `EurliborSwapFixA4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.312 EurliborSwapFixA5Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA5Y:



### 7.312.1 Detailed Description

5-year EurliborSwapFixA index

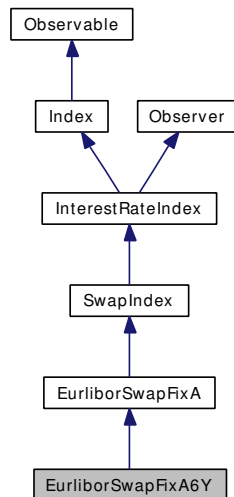
#### Public Member Functions

- `EurliborSwapFixA5Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.313 EurliborSwapFixA6Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA6Y:



### 7.313.1 Detailed Description

6-year EurliborSwapFixA index

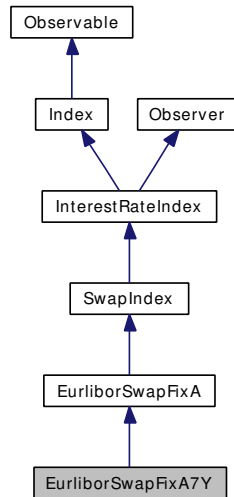
#### Public Member Functions

- `EurliborSwapFixA6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.314 EurliborSwapFixA7Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA7Y:



### 7.314.1 Detailed Description

7-year EurliborSwapFixA index

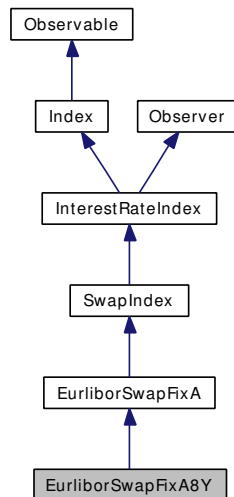
#### Public Member Functions

- EurliborSwapFixA7Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.315 EurliborSwapFixA8Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA8Y:



### 7.315.1 Detailed Description

8-year EurliborSwapFixA index

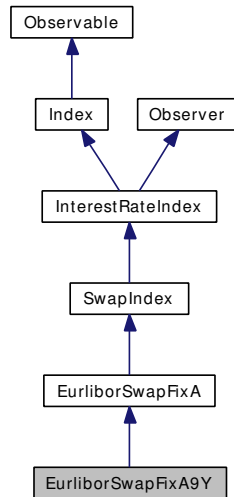
#### Public Member Functions

- `EurliborSwapFixA8Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.316 EurliborSwapFixA9Y Class Reference

```
#include <ql/Indexes/eurliborswapfixa.hpp>
```

Inheritance diagram for EurliborSwapFixA9Y:



### 7.316.1 Detailed Description

9-year EurliborSwapFixA index

#### Public Member Functions

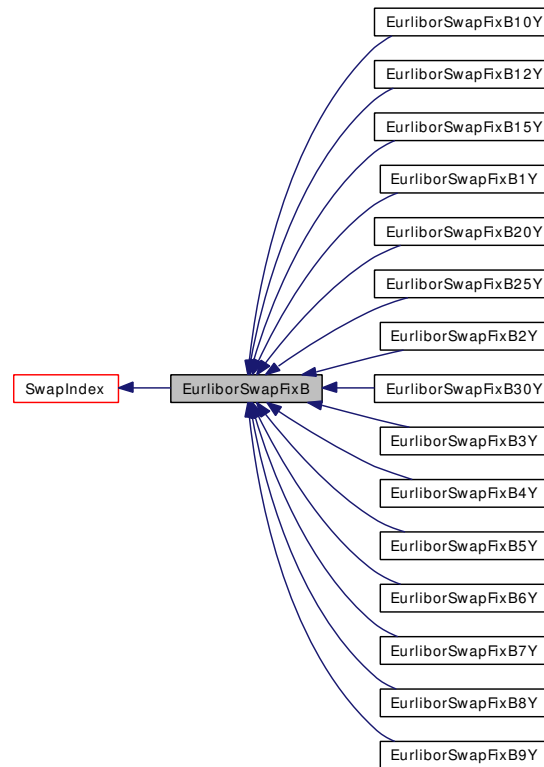
- EurliborSwapFixA9Y (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.317 EurliborSwapFixB Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB:



### 7.317.1 Detailed Description

EurliborSwapFixB index

**EurliborSwapFixB** rate fixed by ISDA in cooperation with Reuters and Intercapital Brokers. The swap index is based on the EuroLibor 6M and is fixed at 11:00AM London. Reuters page ISDAFIX2 or EURSFXLB= Further info can be found at: <http://www.isda.org/fix/isdafix.html>.

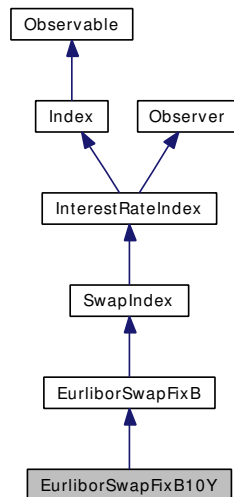
### Public Member Functions

- **EurliborSwapFixB** (*Integer* years, const *Handle*< *YieldTermStructure* > &h=*Handle*< *YieldTermStructure* >())

## 7.318 EurliborSwapFixB10Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB10Y:



### 7.318.1 Detailed Description

10-year EurliborSwapFixB index

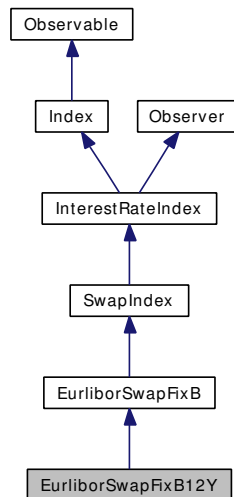
#### Public Member Functions

- EurliborSwapFixB10Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.319 EurliborSwapFixB12Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB12Y:



### 7.319.1 Detailed Description

12-year EurliborSwapFixB index

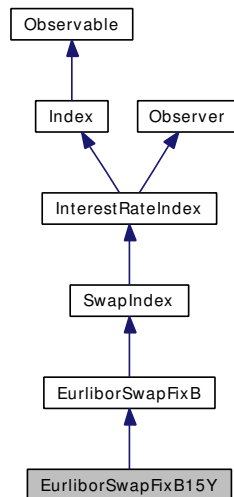
#### Public Member Functions

- `EurliborSwapFixB12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.320 EurliborSwapFixB15Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB15Y:



### 7.320.1 Detailed Description

15-year EurliborSwapFixB index

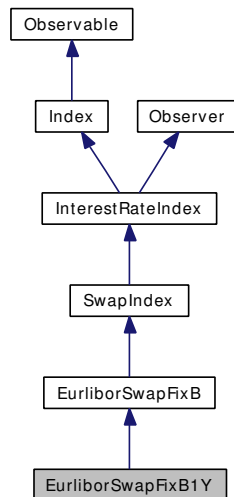
#### Public Member Functions

- EurliborSwapFixB15Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.321 EurliborSwapFixB1Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB1Y:



### 7.321.1 Detailed Description

1-year EurliborSwapFixB index

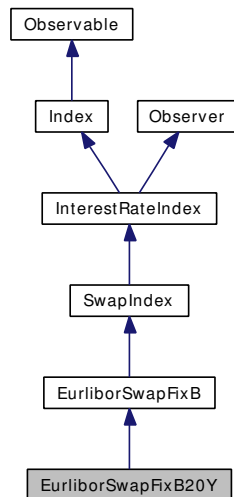
#### Public Member Functions

- `EurliborSwapFixB1Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.322 EurliborSwapFixB20Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB20Y:



### 7.322.1 Detailed Description

20-year EurliborSwapFixB index

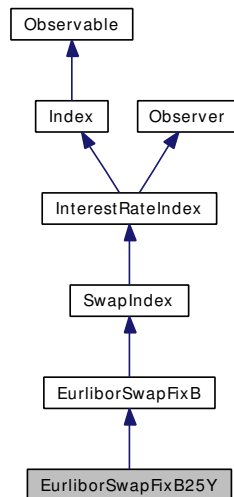
#### Public Member Functions

- `EurliborSwapFixB20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.323 EurliborSwapFixB25Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB25Y:



### 7.323.1 Detailed Description

25-year EurliborSwapFixB index

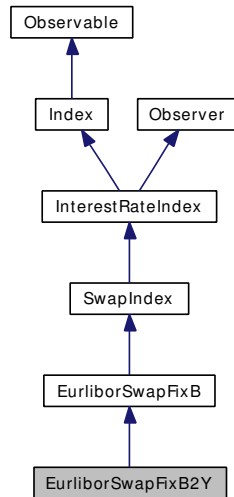
#### Public Member Functions

- `EurliborSwapFixB25Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.324 EurliborSwapFixB2Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB2Y:



### 7.324.1 Detailed Description

2-year EurliborSwapFixB index

#### Public Member Functions

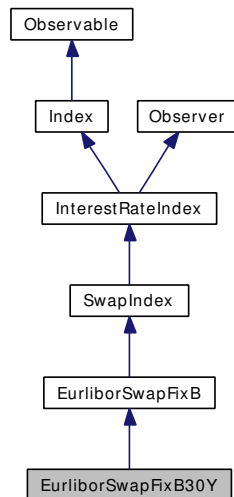
- EurliborSwapFixB2Y (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.325 EurliborSwapFixB30Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB30Y:



### 7.325.1 Detailed Description

30-year EurliborSwapFixB index

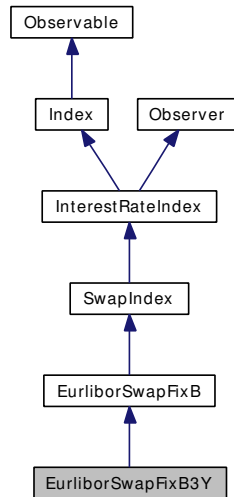
#### Public Member Functions

- `EurliborSwapFixB30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.326 EurliborSwapFixB3Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB3Y:



### 7.326.1 Detailed Description

3-year EurliborSwapFixB index

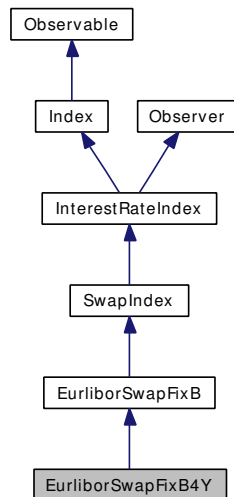
#### Public Member Functions

- EurliborSwapFixB3Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.327 EurliborSwapFixB4Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB4Y:



### 7.327.1 Detailed Description

4-year EurliborSwapFixB index

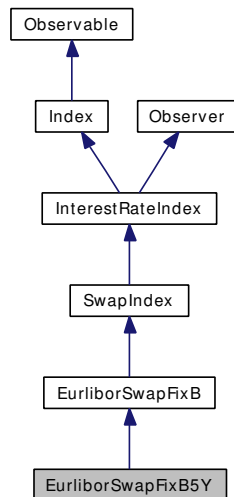
#### Public Member Functions

- `EurliborSwapFixB4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.328 EurliborSwapFixB5Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB5Y:



### 7.328.1 Detailed Description

5-year EurliborSwapFixB index

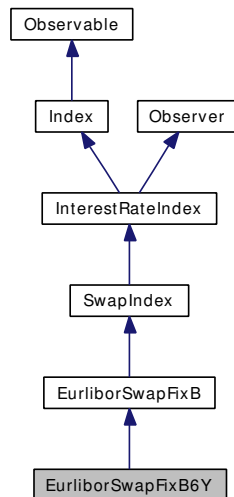
#### Public Member Functions

- EurliborSwapFixB5Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.329 EurliborSwapFixB6Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB6Y:



### 7.329.1 Detailed Description

6-year EurliborSwapFixB index

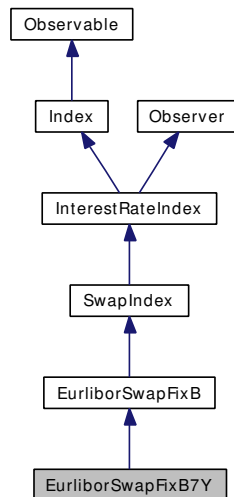
#### Public Member Functions

- `EurliborSwapFixB6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.330 EurliborSwapFixB7Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB7Y:



### 7.330.1 Detailed Description

7-year EurliborSwapFixB index

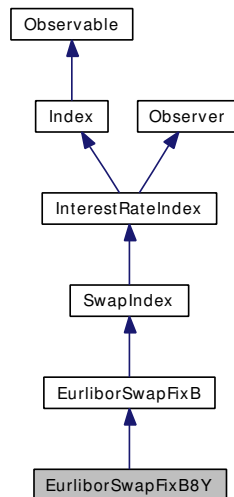
#### Public Member Functions

- EurliborSwapFixB7Y (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.331 EurliborSwapFixB8Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB8Y:



### 7.331.1 Detailed Description

8-year EurliborSwapFixB index

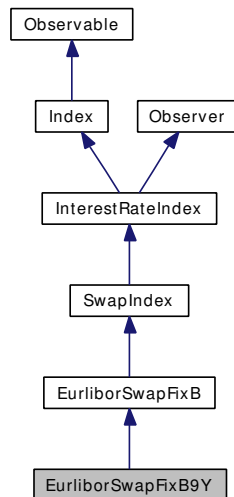
#### Public Member Functions

- `EurliborSwapFixB8Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.332 EurliborSwapFixB9Y Class Reference

```
#include <ql/Indexes/eurliborswapfixb.hpp>
```

Inheritance diagram for EurliborSwapFixB9Y:



### 7.332.1 Detailed Description

9-year EurliborSwapFixB index

#### Public Member Functions

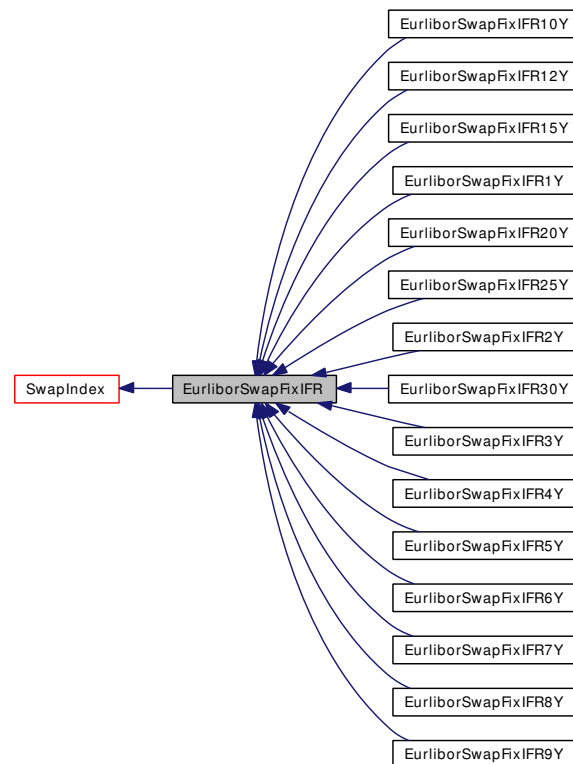
- EurliborSwapFixB9Y (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.333 EurliborSwapFixIFR Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR:



### 7.333.1 Detailed Description

EurliborSwapFixIFR index

EuriborSwapFix index published by IFR Markets and distributed by Reuters page TGM42281 and by Telerate. For more info see <<http://www.ifrmarkets.com>>.

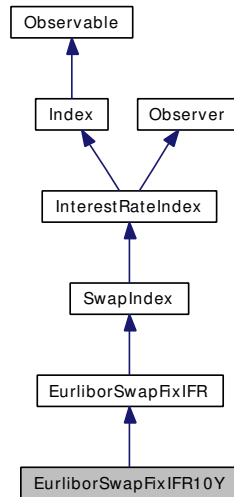
### Public Member Functions

- **EurliborSwapFixIFR** ([Integer](#) years, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.334 EurliborSwapFixIFR10Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR10Y:



### 7.334.1 Detailed Description

10-year EurliborSwapFixIFR index

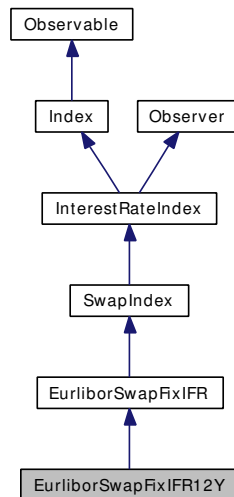
#### Public Member Functions

- `EurliborSwapFixIFR10Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.335 EurliborSwapFixIFR12Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR12Y:



### 7.335.1 Detailed Description

12-year EurliborSwapFixIFR index

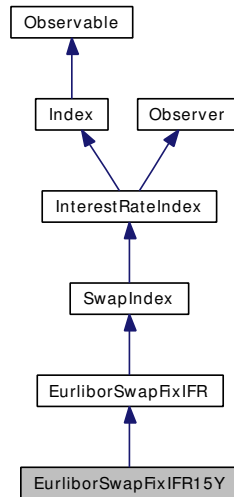
#### Public Member Functions

- `EurliborSwapFixIFR12Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.336 EurliborSwapFixIFR15Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR15Y:



### 7.336.1 Detailed Description

15-year EurliborSwapFixIFR index

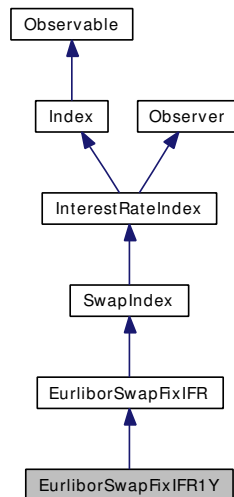
#### Public Member Functions

- `EurliborSwapFixIFR15Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.337 EurliborSwapFixIFR1Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR1Y:



### 7.337.1 Detailed Description

1-year EurliborSwapFixIFR index

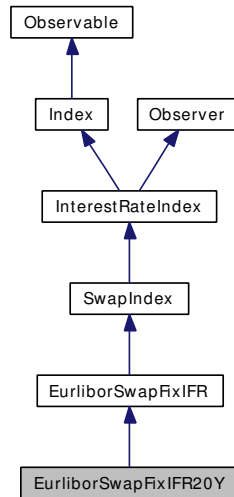
#### Public Member Functions

- `EurliborSwapFixIFR1Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.338 EurliborSwapFixIFR20Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR20Y:



### 7.338.1 Detailed Description

20-year EurliborSwapFixIFR index

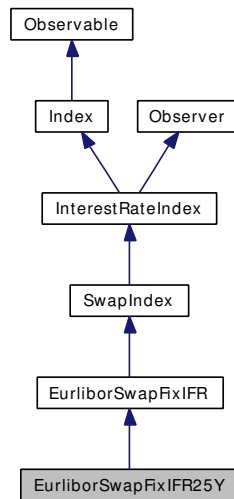
#### Public Member Functions

- `EurliborSwapFixIFR20Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.339 EurliborSwapFixIFR25Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR25Y:



### 7.339.1 Detailed Description

25-year EurliborSwapFixIFR index

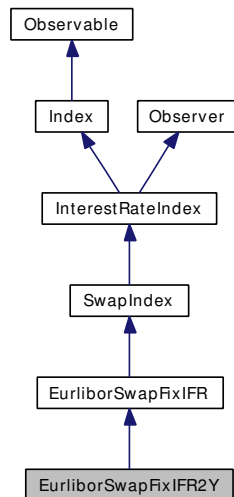
#### Public Member Functions

- `EurliborSwapFixIFR25Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.340 EurliborSwapFixIFR2Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR2Y:



### 7.340.1 Detailed Description

2-year EurliborSwapFixIFR index

#### Public Member Functions

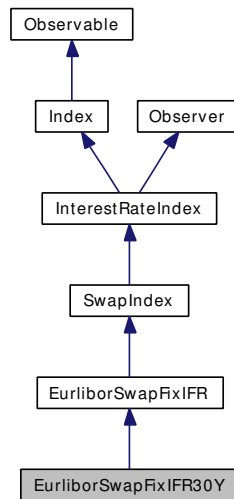
- `EurliborSwapFixIFR2Y` (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.341 EurliborSwapFixIFR30Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR30Y:



### 7.341.1 Detailed Description

30-year EurliborSwapFixIFR index

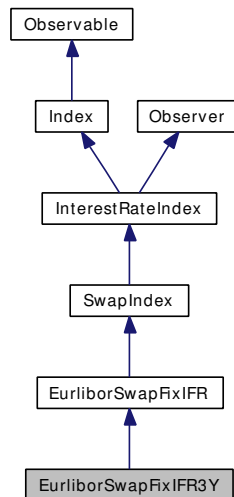
#### Public Member Functions

- `EurliborSwapFixIFR30Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.342 EurliborSwapFixIFR3Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR3Y:



### 7.342.1 Detailed Description

3-year EurliborSwapFixIFR index

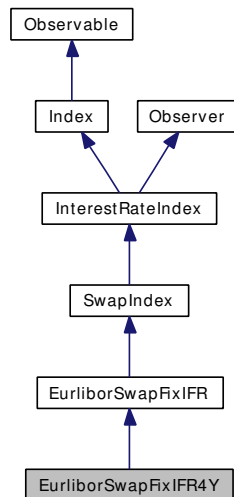
#### Public Member Functions

- `EurliborSwapFixIFR3Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.343 EurliborSwapFixIFR4Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR4Y:



### 7.343.1 Detailed Description

4-year EurliborSwapFixIFR index

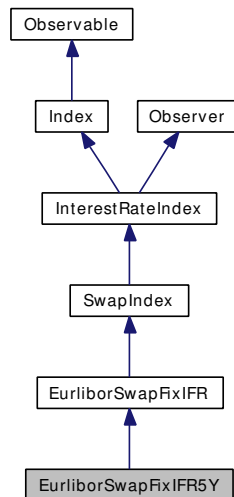
#### Public Member Functions

- `EurliborSwapFixIFR4Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.344 EurliborSwapFixIFR5Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR5Y:



### 7.344.1 Detailed Description

5-year EurliborSwapFixIFR index

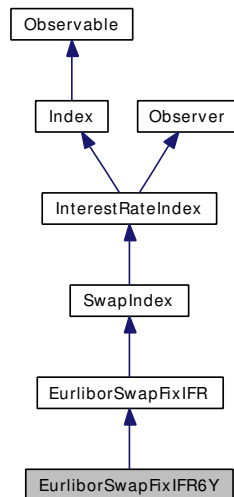
#### Public Member Functions

- `EurliborSwapFixIFR5Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.345 EurliborSwapFixIFR6Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR6Y:



### 7.345.1 Detailed Description

6-year EurliborSwapFixIFR index

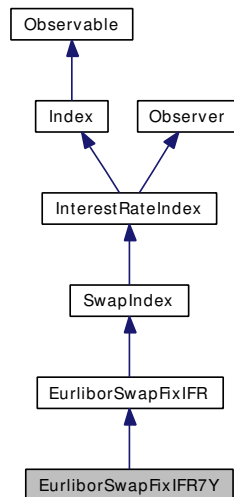
#### Public Member Functions

- `EurliborSwapFixIFR6Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.346 EurliborSwapFixIFR7Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR7Y:



### 7.346.1 Detailed Description

7-year EurliborSwapFixIFR index

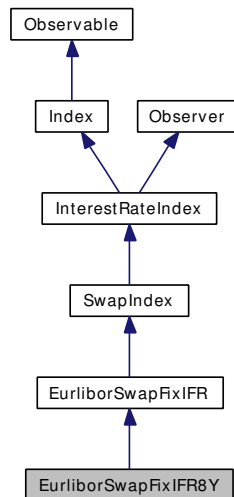
#### Public Member Functions

- `EurliborSwapFixIFR7Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.347 EurliborSwapFixIFR8Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR8Y:



### 7.347.1 Detailed Description

8-year EurliborSwapFixIFR index

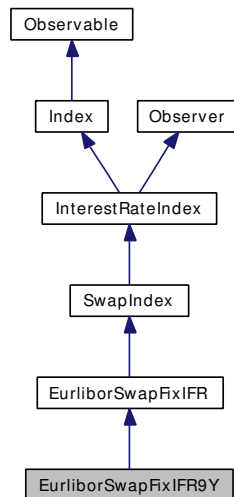
#### Public Member Functions

- `EurliborSwapFixIFR8Y` (const [Handle](#)< [YieldTermStructure](#) > &h)

## 7.348 EurliborSwapFixIFR9Y Class Reference

```
#include <ql/Indexes/eurliborswapfixifr.hpp>
```

Inheritance diagram for EurliborSwapFixIFR9Y:



### 7.348.1 Detailed Description

9-year EurliborSwapFixIFR index

#### Public Member Functions

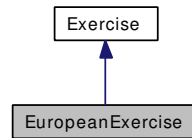
- `EurliborSwapFixIFR9Y` (const [Handle](#)< [YieldTermStructure](#) > &h)



## 7.349 EuropeanExercise Class Reference

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

Inheritance diagram for EuropeanExercise:



### 7.349.1 Detailed Description

European exercise.

A European option can only be exercised at one (expiry) date.

**Examples:**

[ConvertibleBonds.cpp](#), [EquityOption.cpp](#), and [Replication.cpp](#).

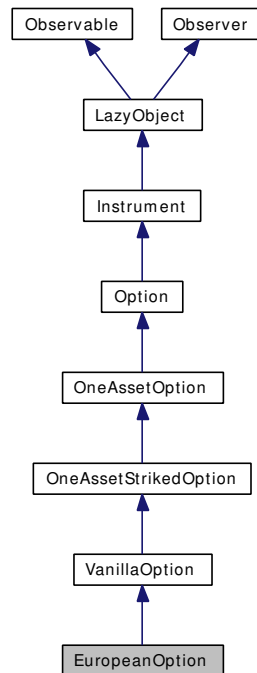
### Public Member Functions

- **EuropeanExercise** (const [Date](#) &date)

## 7.350 EuropeanOption Class Reference

```
#include <ql/Instruments/europeanoption.hpp>
```

Inheritance diagram for EuropeanOption:



### 7.350.1 Detailed Description

European option on a single asset.

**Examples:**

[Replication.cpp](#).

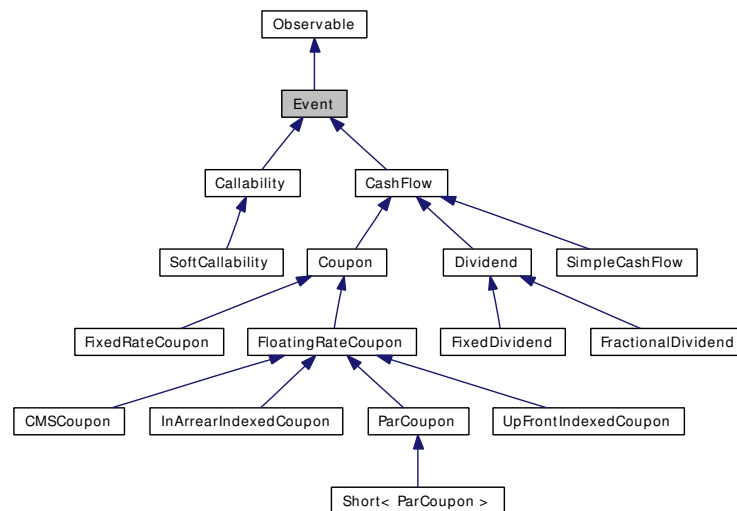
### Public Member Functions

- **EuropeanOption** (const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &, const boost::shared\_ptr< [Exercise](#) > &, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())

## 7.351 Event Class Reference

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

Inheritance diagram for Event:



### 7.351.1 Detailed Description

Base class for event.

This class acts as a base class for the actual event implementations.

### Public Member Functions

#### Event interface

- virtual [Date](#) [date](#) () const=0  
*returns the date at which the event occurs*
- bool [hasOccurred](#) (const [Date](#) &d, bool includeToday=false) const  
*returns true if an event has already occurred before a date*

#### Visitability

- virtual void [accept](#) ([AcyclicVisitor](#) &)

### 7.351.2 Member Function Documentation

#### 7.351.2.1 bool hasOccurred (const [Date](#) & d, bool includeToday = false) const

returns true if an event has already occurred before a date

If `QL_TODAYS_PAYMENT` is true, then a payment event has not occurred if the input date is the same as the event date, and so `includeToday` should be defaulted to true.

This should be the only place in the code that is affected directly by `QL_TODAYS_PAYMENT`

#### Todo

- make `QL_TODAYS_PAYMENT` dynamically configurable?

## 7.352 EvolutionDescription Class Reference

```
#include <ql/MarketModels/evolutiondescription.hpp>
```

### 7.352.1 Detailed Description

This class stores: 1) `evolutionTimes` = the times defining the rates that are to be evolved, 2) `rateTimes` = the times at which the rates need to be known, 3) `relevanceRates` = which rates need to be known at each time. This class is really just a tuple of evolution and rate times

- There will be  $n+1$  rate times expressing payment and reset times of forward rates.
- There will be any number of evolution times.
- We also store which part of the rates are relevant for pricing via relevance rates. The important part for the  $i$ -th step will then range from `relevanceRates[i].first` to `relevanceRates[i].second`. Default values for relevance rates will be 0 and  $n$ .
- Example  $n = 5$ : `|—|—|—|—|—|` (size = 6) `t0 t1 t2 t3 t4 t5 rateTimes f0 f1 f2 f3 f4 forward-Rates d0 d1 d2 d3 d4 d5 discountBonds d0/d0 d1/d0 d2/d0 d3/d0 d4/d0 d5/d0 discountRatios sr0 sr1 sr2 sr3 sr4 coterminalswaps`

### Public Member Functions

- **EvolutionDescription** (const std::vector< Time > &`rateTimes`, const std::vector< Time > &`evolutionTimes`, const std::vector< std::pair< Size, Size > > &`relevanceRates`=std::vector< range >())
- const std::vector< Time > & **rateTimes** () const
- const std::vector< Time > & **rateTaus** () const
- const std::vector< Time > & **evolutionTimes** () const
- const [Matrix](#) & **effectiveStopTime** () const
- const std::vector< Size > & **firstAliveRate** () const
- const std::vector< std::pair< Size, Size > > & **relevanceRates** () const
- Size **numberOfRates** () const
- Size **numberOfSteps** () const

## 7.353 ExchangeRate Class Reference

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

### 7.353.1 Detailed Description

exchange rate between two currencies

#### Tests

application of direct and derived exchange rate is tested against calculations.

### Utility methods

- [Money exchange](#) (const [Money](#) &amount) const  
*apply the exchange rate to a cash amount*
- static [ExchangeRate chain](#) (const [ExchangeRate](#) &r1, const [ExchangeRate](#) &r2)  
*chain two exchange rates*

### Public Types

- enum [Type](#) { [Direct](#), [Derived](#) }

### Public Member Functions

#### Inspectors

- const [Currency](#) & [source](#) () const  
*the source currency.*
- const [Currency](#) & [target](#) () const  
*the target currency.*
- [Type](#) [type](#) () const  
*the type*
- Decimal [rate](#) () const  
*the exchange rate (when available)*

### 7.353.2 Member Enumeration Documentation

#### 7.353.2.1 enum [Type](#)

##### Enumerator:

*Direct* given directly by the user

*Derived* derived from exchange rates between other currencies

### 7.353.3 Constructor & Destructor Documentation

#### 7.353.3.1 `ExchangeRate` (const `Currency` & *source*, const `Currency` & *target*, Decimal *rate*)

the rate  $r$  is given with the convention that a unit of the source is worth  $r$  units of the target.

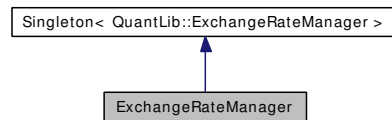
#### 7.353.3.2 `ExchangeRate` (const `Currency` & *source*, const `Currency` & *target*, Decimal *rate*)

the rate  $r$  is given with the convention that a unit of the source is worth  $r$  units of the target.

## 7.354 ExchangeRateManager Class Reference

```
#include <ql/Currencies/exchangeratemanager.hpp>
```

Inheritance diagram for ExchangeRateManager:



### 7.354.1 Detailed Description

exchange-rate repository

#### Tests

lookup of direct, triangulated, and derived exchange rates is tested.

### Public Member Functions

- void **add** (const [ExchangeRate](#) &, const [Date](#) &startDate=Date::minDate(), const [Date](#) &endDate=Date::maxDate())  
*Add an exchange rate.*
- [ExchangeRate](#) **lookup** (const [Currency](#) &source, const [Currency](#) &target, [Date](#) date=[Date](#)(), [ExchangeRate::Type](#) type=[ExchangeRate::Derived](#)) const
- void **clear** ()  
*remove the added exchange rates*

### Friends

- class **Singleton**< [ExchangeRateManager](#) >

### 7.354.2 Member Function Documentation

**7.354.2.1 void add** (const [ExchangeRate](#) &, const [Date](#) & startDate = Date::minDate(), const [Date](#) & endDate = Date::maxDate())

Add an exchange rate.

The given rate is valid between the given dates.

#### Note:

If two rates are given between the same currencies and with overlapping date ranges, the latest one added takes precedence during lookup.



7.354.2.2 **ExchangeRate** lookup (const **Currency** & *source*, const **Currency** & *target*, **Date** *date* = **Date**(), **ExchangeRate::Type** *type* = ExchangeRate::Derived) const

Lookup the exchange rate between two currencies at a given date. If the given type is Direct, only direct exchange rates will be returned if available; if Derived, direct rates are still preferred but derived rates are allowed.

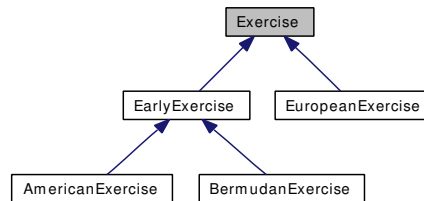
#### Warning

if two or more exchange-rate chains are possible which allow to specify a requested rate, it is unspecified which one is returned.

## 7.355 Exercise Class Reference

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

Inheritance diagram for Exercise:



### 7.355.1 Detailed Description

Base exercise class.

#### Public Types

- enum Type { American, Bermudan, European }

#### Public Member Functions

- Exercise (Type type)
- Type type () const
- Date date (Size index) const
- const std::vector< Date > & dates () const  
*Returns all exercise dates.*
- Date lastDate () const

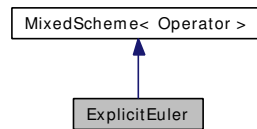
#### Protected Attributes

- std::vector< Date > dates\_
- Type type\_

## 7.356 ExplicitEuler Class Template Reference

```
#include <ql/FiniteDifferences/expliciteuler.hpp>
```

Inheritance diagram for ExplicitEuler:



### 7.356.1 Detailed Description

```
template<class Operator> class QuantLib::ExplicitEuler< Operator >
```

**Forward** Euler scheme for finite difference methods.

See sect. [Finite-differences framework](#) 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 ... array_type;

// 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
array_type applyTo(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator-(const Operator&, const Operator&);

```

#### Todo

add Richardson extrapolation

### Public Types

- `typedef OperatorTraits< Operator > traits`
- `typedef traits::operator_type operator_type`
- `typedef traits::array_type array_type`
- `typedef traits::bc_type bc_type`
- `typedef traits::bc_set bc_set`
- `typedef traits::condition_type condition_type`

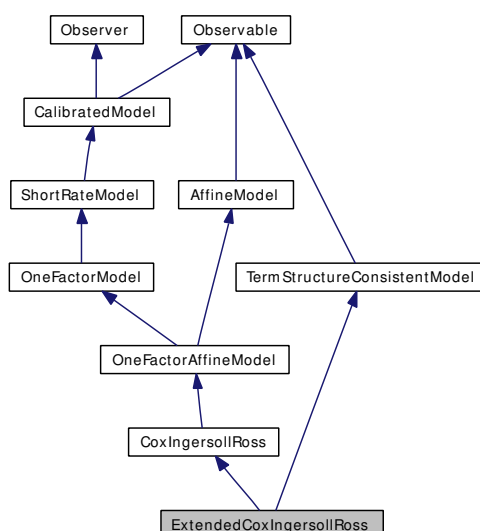
## Public Member Functions

- **ExplicitEuler** (const operator\_type &L, const std::vector< boost::shared\_ptr< [bc\\_type](#) > > &bcs)

## 7.357 ExtendedCoxIngersollRoss Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp>
```

Inheritance diagram for ExtendedCoxIngersollRoss:



### 7.357.1 Detailed Description

Extended Cox-Ingersoll-Ross model class.

This class implements the extended Cox-Ingersoll-Ross model defined by

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

#### Bug

this class was not tested enough to guarantee its functionality.

### Public Member Functions

- **ExtendedCoxIngersollRoss** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, Real theta=0.1, Real k=0.1, Real sigma=0.1, Real x0=0.05)
- boost::shared\_ptr< [NumericalMethod](#) > **tree** (const [TimeGrid](#) &grid) const  
*Return by default a trinomial recombining tree.*
- boost::shared\_ptr< [ShortRateDynamics](#) > **dynamics** () const  
*returns the short-rate dynamics*
- Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bond-Maturity) const

## Protected Member Functions

- void **generateArguments** ()
- Real **A** (Time t, Time T) const

## Classes

- class [Dynamics](#)  
*Short-rate dynamics in the extended Cox-Ingersoll-Ross model.*
- class [FittingParameter](#)  
*Analytical term-structure fitting parameter  $\varphi(t)$ .*

## 7.358 ExtendedCoxIngersollRoss::Dynamics Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp>
```

### 7.358.1 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.

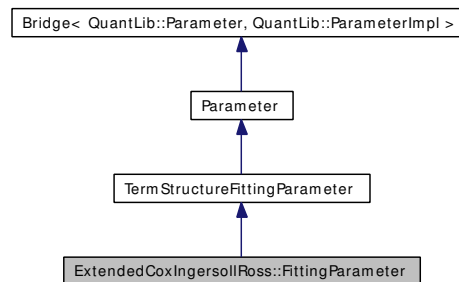
### Public Member Functions

- **Dynamics** (const [Parameter](#) &phi, [Real](#) theta, [Real](#) k, [Real](#) sigma, [Real](#) x0)
- virtual [Real](#) **variable** ([Time](#) t, [Rate](#) r) const
- virtual [Real](#) **shortRate** ([Time](#) t, [Real](#) y) const

## 7.359 ExtendedCoxIngersollRoss::FittingParameter Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp>
```

Inheritance diagram for ExtendedCoxIngersollRoss::FittingParameter:



### 7.359.1 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

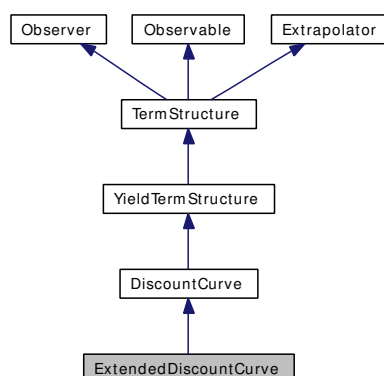
- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) theta, [Real](#) k, [Real](#) sigma, [Real](#) x0)



## 7.360 ExtendedDiscountCurve Class Reference

```
#include <ql/TermStructures/extendeddiscountcurve.hpp>
```

Inheritance diagram for ExtendedDiscountCurve:



### 7.360.1 Detailed Description

Term structure based on loglinear interpolation of discount factors.

Loglinear interpolation guarantees piecewise constant forward rates.

Rates are assumed to be annual continuous compounding.

### Public Member Functions

- **ExtendedDiscountCurve** (const std::vector< [Date](#) > &dates, const std::vector< [DiscountFactor](#) > &dfs, const [Calendar](#) &calendar, const [BusinessDayConvention](#) conv, const [DayCounter](#) &dayCounter)
- [Calendar](#) **calendar** () const  
*the calendar used for reference date calculation*
- [BusinessDayConvention](#) **businessDayConvention** () const
- void **update** ()
- [Rate](#) **compoundForward** (const [Date](#) &d1, Integer f, bool extrapolate=false) const
- [Rate](#) **compoundForward** (Time t1, Integer f, bool extrapolate=false) const

### Protected Member Functions

- [Rate](#) **compoundForwardImpl** (Time, Integer) const
- [Rate](#) **zeroYieldImpl** (Time) const
- void **calibrateNodes** () const
- boost::shared\_ptr< [CompoundForward](#) > **reversebootstrap** (Integer) const
- boost::shared\_ptr< [CompoundForward](#) > **forwardCurve** (Integer) const

## 7.360.2 Member Function Documentation

### 7.360.2.1 `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 [TermStructure](#).

### 7.360.2.2 `Rate compoundForwardImpl (Time, Integer) const` [protected]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

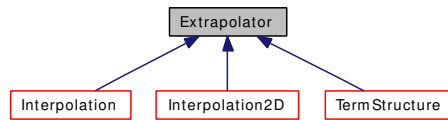
### 7.360.2.3 `Rate zeroYieldImpl (Time) const` [protected]

Returns the zero yield rate for the given date calculating it from the discount.

## 7.361 Extrapolator Class Reference

```
#include <ql/Math/extrapolation.hpp>
```

Inheritance diagram for Extrapolator:



### 7.361.1 Detailed Description

base class for classes possibly allowing extrapolation

#### Public Member Functions

##### modifiers

- void [enableExtrapolation](#) (bool b=true)  
*enable extrapolation in subsequent calls*
- void [disableExtrapolation](#) (bool b=true)  
*disable extrapolation in subsequent calls*

##### inspectors

- bool [allowsExtrapolation](#) () const  
*tells whether extrapolation is enabled*

## 7.362 Factorial Class Reference

```
#include <ql/Math/factorial.hpp>
```

### 7.362.1 Detailed Description

Factorial numbers calculator

#### Tests

the correctness of the returned value is tested by checking it against numerical calculations.

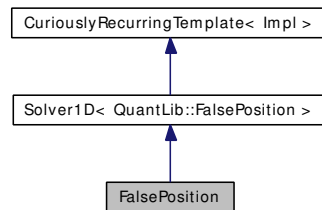
### Static Public Member Functions

- static [Real](#) `get` (Natural n)
- static [Real](#) `ln` (Natural n)

## 7.363 FalsePosition Class Reference

```
#include <ql/Solvers1D/falseposition.hpp>
```

Inheritance diagram for FalsePosition:



### 7.363.1 Detailed Description

False position 1-D solver.

#### Tests

the correctness of the returned values is tested by checking them against known good results.

### Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

## 7.364 FaureRsg Class Reference

```
#include <ql/RandomNumbers/faurersg.hpp>
```

### 7.364.1 Detailed Description

Faure low-discrepancy sequence generator.

It is based on existing Fortran and C algorithms to calculate pascal matrix and gray transforms.

1. E. Thiernard Economic generation of low-discrepancy sequences with a b-ary gray code.
2. Algorithms 659, 647. <http://www.netlib.org/toms/647>,  
<http://www.netlib.org/toms/659>

#### Tests

the correctness of the returned values is tested by reproducing known good values.

### Public Types

- typedef [Sample](#)< [Array](#) > **sample\_type**

### Public Member Functions

- **FaureRsg** (Size dimensionality)
- const std::vector< long int > & **nextIntSequence** () const
- const std::vector< long int > & **lastIntSequence** () const
- const [sample\\_type](#) & **nextSequence** () const
- const [sample\\_type](#) & **lastSequence** () const
- Size **dimension** () const

## 7.365 FDAmericanCondition Class Template Reference

```
#include <ql/PricingEngines/Vanilla/fdconditions.hpp>
```

### 7.365.1 Detailed Description

```
template<typename baseEngine> class QuantLib::FDAmericanCondition< baseEngine >
```

Generate engine with specific conditions

### Public Member Functions

- **FDAmericanCondition** (Size timeSteps=100, Size gridPoints=100, bool time-Dependent=false)

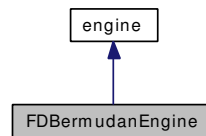
### Protected Member Functions

- void **initializeStepCondition** () const

## 7.366 FDBermudanEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/fdbermudanengine.hpp>
```

Inheritance diagram for FDBermudanEngine:



### 7.366.1 Detailed Description

Finite-differences Bermudan engine.

**Examples:**

[EquityOption.cpp](#).

### Public Member Functions

- `FDBermudanEngine` (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)
- void `calculate` () const

### Protected Member Functions

- void `initializeStepCondition` () const
- void `executeIntermediateStep` (Size) const

### Protected Attributes

- Real `extraTermInBermudan`



## 7.367 FDDividendEngineMerton73 Class Reference

```
#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>
```

### 7.367.1 Detailed Description

Finite-differences pricing engine for dividend options using.

### Public Member Functions

- **FDDividendEngineMerton73** (Size timeSteps=100, Size gridPoints=100, bool time-Dependent=false)

## 7.368 FDDividendEngineShiftScale Class Reference

```
#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>
```

### 7.368.1 Detailed Description

Finite-differences pricing engine for dividend options using.

#### Public Member Functions

- **FDDividendEngineShiftScale** (Size timeSteps=100, Size gridPoints=100, bool time-Dependent=false)

## 7.369 FDEuropeanEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/fdeuropeanengine.hpp>
```

### 7.369.1 Detailed Description

Pricing engine for European options using finite-differences.

#### Tests

the correctness of the returned value is tested by checking it against analytic results.

#### Examples:

[EquityOption.cpp](#).

### Public Member Functions

- **FDEuropeanEngine** (Size timeSteps=100, Size gridPoints=100, bool timeDependent=false)

## 7.370 FDStepConditionEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/fdstepconditionengine.hpp>
```

### 7.370.1 Detailed Description

Finite-differences pricing engine for American-style vanilla options.

#### Public Member Functions

- **FDStepConditionEngine** (Size timeSteps, Size gridPoints, bool timeDependent=false)

#### Protected Member Functions

- virtual void **initializeStepCondition** () const=0
- virtual void **calculate** ([Results](#) \*result) const

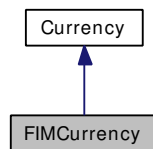
#### Protected Attributes

- boost::shared\_ptr< [StandardStepCondition](#) > **stepCondition\_**
- [SampledCurve](#) **prices\_**
- [TridiagonalOperator](#) **controlOperator\_**
- std::vector< boost::shared\_ptr< bc\_type > > **controlBCs\_**
- [SampledCurve](#) **controlPrices\_**

## 7.371 FIMCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for FIMCurrency:



### 7.371.1 Detailed Description

Finnish markka.

The ISO three-letter code was FIM; the numeric code was 246. It was divided in 100 penniä.

Obsoleted by the Euro since 1999.

## 7.372 FiniteDifferenceModel Class Template Reference

```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
```

### 7.372.1 Detailed Description

```
template<class Evolver> class QuantLib::FiniteDifferenceModel< Evolver >
```

Generic finite difference model.

### Public Types

- typedef Evolver::traits **traits**
- typedef traits::operator\_type **operator\_type**
- typedef traits::array\_type **array\_type**
- typedef traits::bc\_set **bc\_set**
- typedef traits::condition\_type **condition\_type**

### Public Member Functions

- **FiniteDifferenceModel** (const operator\_type &L, const bc\_set &bcs, const std::vector< Time > &stoppingTimes=std::vector< Time >())
- **FiniteDifferenceModel** (const Evolver &evolver, const std::vector< Time > &stoppingTimes=std::vector< Time >())
- const Evolver & **evolver** () const
- void **rollback** (array\_type &a, Time from, Time to, Size steps)
- void **rollback** (array\_type &a, Time from, Time to, Size steps, const condition\_type &condition)

### 7.372.2 Member Function Documentation

#### 7.372.2.1 void rollback (array\_type & a, Time from, Time to, Size steps)

solves the problem between the given times.

#### Warning

being this a rollback, from must be a later time than to.

#### 7.372.2.2 void rollback (array\_type & a, Time from, Time to, Size steps, const condition\_type & condition)

solves the problem between the given times, applying a condition at every step.

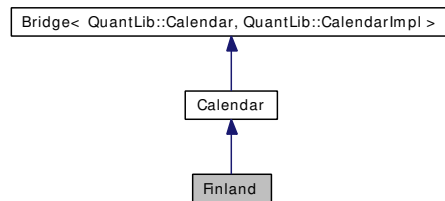
#### Warning

being this a rollback, from must be a later time than to.

## 7.373 Finland Class Reference

```
#include <ql/Calendars/finland.hpp>
```

Inheritance diagram for Finland:



### 7.373.1 Detailed Description

Finnish calendar.

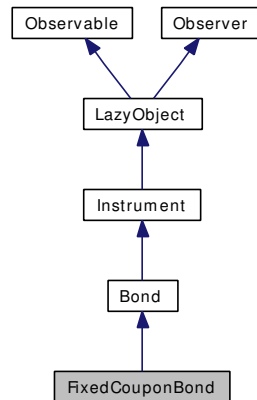
Holidays:

- 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

## 7.374 FixedCouponBond Class Reference

```
#include <ql/Instruments/fixedcouponbond.hpp>
```

Inheritance diagram for FixedCouponBond:



### 7.374.1 Detailed Description

fixed-coupon bond

#### Tests

calculations are tested by checking results against cached values.

#### Examples:

[Repo.cpp](#).

### Public Member Functions

- **FixedCouponBond** ([Real](#) faceAmount, const [Date](#) &issueDate, const [Date](#) &datedDate, const [Date](#) &maturityDate, [Integer](#) settlementDays, const std::vector< [Rate](#) > &coupons, [Frequency](#) couponFrequency, const [Calendar](#) &calendar, const [DayCounter](#) &dayCounter, [BusinessDayConvention](#) accrualConvention=Following, [BusinessDayConvention](#) paymentConvention=Following, [Real](#) redemption=100.0, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >(), const [Date](#) &stub=[Date](#)(), bool fromEnd=true, bool longFinal=false)
- **FixedCouponBond** (const [Date](#) &issueDate, const [Date](#) &datedDate, const [Date](#) &maturityDate, [Integer](#) settlementDays, const std::vector< [Rate](#) > &coupons, [Frequency](#) couponFrequency, const [Calendar](#) &calendar, const [DayCounter](#) &dayCounter, [BusinessDayConvention](#) accrualConvention=Following, [BusinessDayConvention](#) paymentConvention=Following, [Real](#) redemption=100.0, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >(), const [Date](#) &stub=[Date](#)(), bool fromEnd=true, bool longFinal=false)



## 7.374.2 Constructor & Destructor Documentation

7.374.2.1 **FixedCouponBond** (const **Date** & *issueDate*, const **Date** & *datedDate*, const **Date** & *maturityDate*, **Integer** *settlementDays*, const std::vector< **Rate** > & *coupons*, **Frequency** *couponFrequency*, const **Calendar** & *calendar*, const **DayCounter** & *dayCounter*, **BusinessDayConvention** *accrualConvention* = Following, **BusinessDayConvention** *paymentConvention* = Following, **Real** *redemption* = 100.0, const **Handle**< **YieldTermStructure** > & *discountCurve* = **Handle**< **YieldTermStructure** >(), const **Date** & *stub* = **Date**(), bool *fromEnd* = true, bool *longFinal* = false)

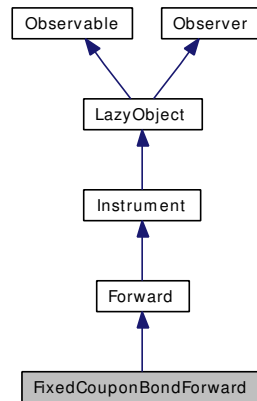
### Deprecated

use constructor with face amount instead

## 7.375 FixedCouponBondForward Class Reference

```
#include <ql/Instruments/fixedcouponbondforward.hpp>
```

Inheritance diagram for FixedCouponBondForward:



### 7.375.1 Detailed Description

forward contract on a fixed-coupon bond

1. `valueDate` refers to the settlement date of the bond forward contract. `maturityDate` is the delivery (or repurchase) date for the underlying bond (not the bond's maturity date).

2. Relevant formulas used in the calculations ( $P$  refers to a price):

a.  $P_{CleanFwd}(t) = P_{DirtyFwd}(t) - AI(t = deliveryDate)$  where  $AI$  refers to the accrued interest on the underlying bond.

b.  $P_{DirtyFwd}(t) = \frac{P_{DirtySpot}(t) - SpotIncome(t)}{discountCurve \rightarrow discount(t=deliveryDate)}$

c.  $SpotIncome(t) = \sum_i (CF_i \times incomeDiscountCurve \rightarrow discount(t_i))$  where  $CF_i$  represents the  $i$ th bond cash flow (coupon payment) associated with the underlying bond falling between the `settlementDate` and the `deliveryDate`. (Note the two different discount curves used in b. and c.)

**Example:** [valuation of a repo on a fixed-coupon bond](#)

#### Todo

Add preconditions and tests

#### Todo

Create switch- if coupon goes to seller is toggled on, don't consider income in the  $P_{DirtyFwd}(t)$  calculation.

#### Todo

Verify this works when the underlying is paper (in which case ignore all AI.)

#### Warning

This class still needs to be rigorously tested

Examples:

[Repo.cpp](#).

## Public Member Functions

### Calculations

- [Real forwardPrice](#) () const  
*(dirty) forward bond price*
- [Real cleanForwardPrice](#) () const  
*(dirty) forward bond price minus accrued on bond at delivery*
- [Real spotIncome](#) (const [Handle](#)< [YieldTermStructure](#) > &incomeDiscountCurve) const  
*NPV of bond coupons discounted using incomeDiscountCurve.*
- [Real spotValue](#) () const  
*NPV of underlying bond.*

## Protected Member Functions

- void [performCalculations](#) () const

## Protected Attributes

- boost::shared\_ptr< [FixedCouponBond](#) > [fixedCouponBond\\_](#)

## 7.375.2 Constructor & Destructor Documentation

- 7.375.2.1 [FixedCouponBondForward](#) (const [Date](#) & *valueDate*, const [Date](#) & *maturityDate*, [Position::Type](#) *type*, [Real](#) *strike*, [Integer](#) *settlementDays*, const [DayCounter](#) & *dayCount*, const [Calendar](#) & *calendar*, [BusinessDayConvention](#) *businessDayConvention*, const boost::shared\_ptr< [FixedCouponBond](#) > & *fixedCouponBond*, const [Handle](#)< [YieldTermStructure](#) > & *discountCurve* = [Handle](#)< [YieldTermStructure](#) >(), const [Handle](#)< [YieldTermStructure](#) > & *incomeDiscountCurve* = [Handle](#)< [YieldTermStructure](#) >())

If strike is given in the constructor, can calculate the NPV of the contract via [NPV\(\)](#).

If strike/forward price is desired, it can be obtained via [forwardPrice\(\)](#). In this case, the strike variable in the constructor is irrelevant and will be ignored.

**7.375.2.2 FixedCouponBondForward** (const [Date](#) & *valueDate*, const [Date](#) & *maturityDate*, [Position::Type](#) *type*, [Real](#) *strike*, [Integer](#) *settlementDays*, const [DayCounter](#) & *dayCount*, const [Calendar](#) & *calendar*, [BusinessDayConvention](#) *businessDayConvention*, const boost::shared\_ptr< [FixedCouponBond](#) > & *fixedCouponBond*, const [Handle](#)< [YieldTermStructure](#) > & *discountCurve* = [Handle](#)< [YieldTermStructure](#) >(), const [Handle](#)< [YieldTermStructure](#) > & *incomeDiscountCurve* = [Handle](#)< [YieldTermStructure](#) >())

If strike is given in the constructor, can calculate the NPV of the contract via [NPV\(\)](#).

If strike/forward price is desired, it can be obtained via [forwardPrice\(\)](#). In this case, the strike variable in the constructor is irrelevant and will be ignored.

### 7.375.3 Member Function Documentation

**7.375.3.1 Real spotIncome** (const [Handle](#)< [YieldTermStructure](#) > & *incomeDiscountCurve*)  
const [virtual]

NPV of bond coupons discounted using *incomeDiscountCurve*.

Here only coupons between max(evaluation date, settlement date) and maturity date of bond forward contract are considered income.

Implements [Forward](#).

**7.375.3.2 void performCalculations ()** const [protected, virtual]

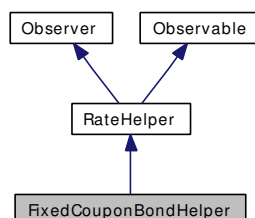
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Forward](#).

## 7.376 FixedCouponBondHelper Class Reference

```
#include <ql/TermStructures/bondhelpers.hpp>
```

Inheritance diagram for FixedCouponBondHelper:



### 7.376.1 Detailed Description

fixed-coupon bond helper

#### Warning

This class assumes that the reference date does not change between calls of [setTermStructure\(\)](#).

### Public Member Functions

- **FixedCouponBondHelper** (const [Handle](#)< [Quote](#) > &cleanPrice, const [Date](#) &issueDate, const [Date](#) &datedDate, const [Date](#) &maturityDate, Integer settlementDays, const std::vector< [Rate](#) > &coupons, [Frequency](#) frequency, const [Calendar](#) &calendar, const [DayCounter](#) &dayCounter, [BusinessDayConvention](#) accrualConvention=Following, [BusinessDayConvention](#) paymentConvention=Following, Real redemption=100.0, const [Date](#) &stub=[Date](#)(), bool fromEnd=true)
- Real **impliedQuote** () const
- [Date](#) **latestDate** () const  
*latest relevant date*
- void **setTermStructure** ([YieldTermStructure](#) \*)  
*sets the term structure to be used for pricing*

### Protected Attributes

- [Date](#) **issueDate\_**
- [Date](#) **datedDate\_**
- [Date](#) **maturityDate\_**
- Integer **settlementDays\_**
- std::vector< [Rate](#) > **coupons\_**
- [Frequency](#) **frequency\_**
- [DayCounter](#) **dayCounter\_**
- [Calendar](#) **calendar\_**
- [BusinessDayConvention](#) **accrualConvention\_**

- [BusinessDayConvention](#) paymentConvention\_
- Real redemption\_
- [Date](#) stub\_
- bool fromEnd\_
- [Date](#) settlement\_
- [Date](#) latestDate\_
- boost::shared\_ptr< [FixedCouponBond](#) > bond\_
- [Handle](#)< [YieldTermStructure](#) > termStructureHandle\_

## 7.376.2 Member Function Documentation

### 7.376.2.1 [Date](#) latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Reimplemented from [RateHelper](#).

### 7.376.2.2 void setTermStructure ([YieldTermStructure](#) \*) [virtual]

sets the term structure to be used for pricing

#### [Warning](#)

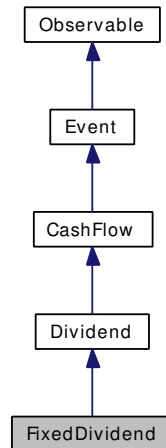
Being a pointer and not a shared\_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

## 7.377 FixedDividend Class Reference

```
#include <ql/CashFlows/dividend.hpp>
```

Inheritance diagram for FixedDividend:



### 7.377.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

**Examples:**

[ConvertibleBonds.cpp](#).

### Public Member Functions

- **FixedDividend** (Real amount, const [Date](#) &date)

#### Dividend interface

- virtual Real [amount](#) () const  
*returns the amount of the cash flow*
- virtual Real **amount** (Real) const

### Protected Attributes

- Real **amount\_**

### 7.377.2 Member Function Documentation

#### 7.377.2.1 virtual Real 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.

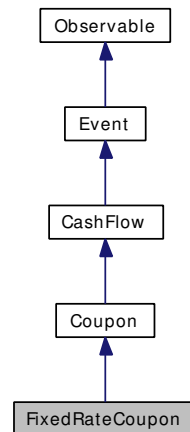
Implements [Dividend](#).



## 7.378 FixedRateCoupon Class Reference

```
#include <ql/CashFlows/fixedratecoupon.hpp>
```

Inheritance diagram for FixedRateCoupon:



### 7.378.1 Detailed Description

Coupon paying a fixed interest rate

#### Public Member Functions

- **FixedRateCoupon** ([Real](#) 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

- [Real](#) **amount** () const  
*returns the amount of the cash flow*

#### Coupon interface

- Rate **rate** () const  
*accrued rate*
- [DayCounter](#) **dayCounter** () const  
*day counter for accrual calculation*
- [Real](#) **accruedAmount** (const [Date](#) &) const  
*accrued amount at the given date*

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.378.2 Member Function Documentation

### 7.378.2.1 [Real](#) amount () const [virtual]

returns the amount of the cash flow

**Note:**

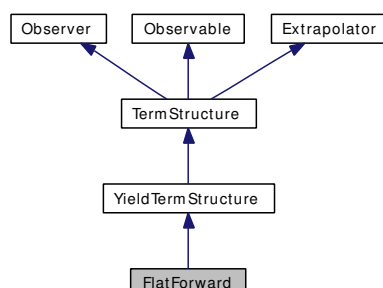
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

## 7.379 FlatForward Class Reference

```
#include <ql/TermStructures/flatforward.hpp>
```

Inheritance diagram for FlatForward:



### 7.379.1 Detailed Description

Flat interest-rate curve.

**Examples:**

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), [Replication.cpp](#), and [Repo.cpp](#).

### Public Member Functions

- **FlatForward** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &forward, const [DayCounter](#) &dayCounter, Compounding compounding=Continuous, [Frequency](#) frequency=Annual)
- **FlatForward** (const [Date](#) &referenceDate, Rate forward, const [DayCounter](#) &dayCounter, Compounding compounding=Continuous, [Frequency](#) frequency=Annual)
- **FlatForward** (Integer settlementDays, const [Calendar](#) &calendar, const [Handle](#)< [Quote](#) > &forward, const [DayCounter](#) &dayCounter, Compounding compounding=Continuous, [Frequency](#) frequency=Annual)
- **FlatForward** (Integer settlementDays, const [Calendar](#) &calendar, Rate forward, const [DayCounter](#) &dayCounter, Compounding compounding=Continuous, [Frequency](#) frequency=Annual)
- [DayCounter](#) dayCounter () const  
*the day counter used for date/time conversion*
- Compounding **compounding** () const
- [Frequency](#) **compoundingFrequency** () const
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- Time **maxTime** () const  
*the latest time for which the curve can return values*
- void **update** ()

## 7.379.2 Member Function Documentation

### 7.379.2.1 `void update ()` [virtual]

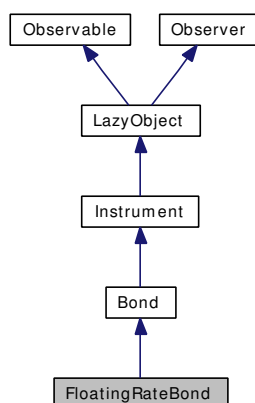
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

## 7.380 FloatingRateBond Class Reference

```
#include <ql/Instruments/floatingratebond.hpp>
```

Inheritance diagram for FloatingRateBond:



### 7.380.1 Detailed Description

floating-rate bond

#### Tests

calculations are tested by checking results against cached values.

### Public Member Functions

- **FloatingRateBond** ([Real](#) faceAmount, const [Date](#) &issueDate, const [Date](#) &datedDate, const [Date](#) &maturityDate, [Integer](#) settlementDays, const boost::shared\_ptr< [Xibor](#) > &index, [Integer](#) fixingDays, const std::vector< [Real](#) > &gearings, const std::vector< [Spread](#) > &spreads, [Frequency](#) couponFrequency, const [Calendar](#) &calendar, const [DayCounter](#) &dayCounter, [BusinessDayConvention](#) accrualConvention=Following, [BusinessDayConvention](#) paymentConvention=Following, [Real](#) redemption=100.0, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >(), const [Date](#) &stub=[Date](#)(), bool fromEnd=true)
- **FloatingRateBond** (const [Date](#) &issueDate, const [Date](#) &datedDate, const [Date](#) &maturityDate, [Integer](#) settlementDays, const boost::shared\_ptr< [Xibor](#) > &index, [Integer](#) fixingDays, const std::vector< [Real](#) > &gearings, const std::vector< [Spread](#) > &spreads, [Frequency](#) couponFrequency, const [Calendar](#) &calendar, const [DayCounter](#) &dayCounter, [BusinessDayConvention](#) accrualConvention=Following, [BusinessDayConvention](#) paymentConvention=Following, [Real](#) redemption=100.0, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >(), const [Date](#) &stub=[Date](#)(), bool fromEnd=true)

## 7.380.2 Constructor & Destructor Documentation

7.380.2.1 **FloatingRateBond** (const **Date** & *issueDate*, const **Date** & *datedDate*, const **Date** & *maturityDate*, **Integer** *settlementDays*, const boost::shared\_ptr< **Xibor** > & *index*, **Integer** *fixingDays*, const std::vector< **Real** > & *gearings*, const std::vector< **Spread** > & *spreads*, **Frequency** *couponFrequency*, const **Calendar** & *calendar*, const **DayCounter** & *dayCounter*, **BusinessDayConvention** *accrualConvention* = Following, **BusinessDayConvention** *paymentConvention* = Following, **Real** *redemption* = 100.0, const **Handle**< **YieldTermStructure** > & *discountCurve* = **Handle**< **YieldTermStructure** >(), const **Date** & *stub* = **Date**(), bool *fromEnd* = true)

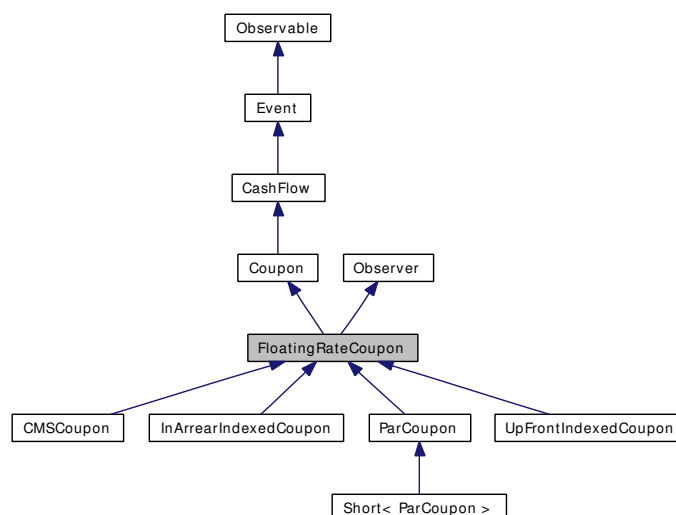
### Deprecated

use constructor with face amount instead

## 7.381 FloatingRateCoupon Class Reference

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

Inheritance diagram for FloatingRateCoupon:



### 7.381.1 Detailed Description

Coupon paying a variable index-based rate

#### 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.

#### Todo

add gearing unit test

### Public Member Functions

- **FloatingRateCoupon** (const [Date](#) &paymentDate, const Real nominal, const [Date](#) &startDate, const [Date](#) &endDate, const Integer fixingDays, const boost::shared\_ptr<[InterestRateIndex](#)> &index, const Real gearing=1.0, const Spread spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())

#### Coupon interface

- virtual Rate [rate](#) () const  
*accrued rate*
- Real [amount](#) () const  
*returns the amount of the cash flow*

- Real `accruedAmount` (const `Date` &) const  
*accrued amount at the given date*
- `DayCounter` `dayCounter` () const  
*day counter for accrual calculation*

### Inspectors

- const boost::shared\_ptr< `InterestRateIndex` > & `index` () const  
*floating index*
- Integer `fixingDays` () const  
*fixing days*
- virtual `Date` `fixingDate` () const  
*fixing date*
- Real `gearing` () const  
*index gearing, i.e. multiplicative coefficient for the index*
- Rate `indexFixing` () const  
*fixing of the underlying index*
- Rate `convexityAdjustment` () const  
*convexity adjustment*
- Rate `adjustedFixing` () const  
*convexity-adjusted fixing*
- Spread `spread` () const  
*spread paid over the fixing of the underlying index*

### Observer interface

- void `update` ()

### Visitability

- virtual void `accept` (`AcyclicVisitor` &)

### Protected Member Functions

- virtual Rate `convexityAdjustmentImpl` (Rate fixing) const  
*convexity adjustment for the given index fixing*

### Protected Attributes

- boost::shared\_ptr< `InterestRateIndex` > `index_`
- `DayCounter` `dayCounter_`
- Integer `fixingDays_`
- Real `gearing_`
- Spread `spread_`



## 7.381.2 Member Function Documentation

### 7.381.2.1 Real 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](#).

### 7.381.2.2 void update () [virtual]

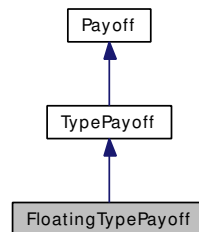
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.382 FloatingTypePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for FloatingTypePayoff:



### 7.382.1 Detailed Description

[Payoff](#) based on a floating strike.

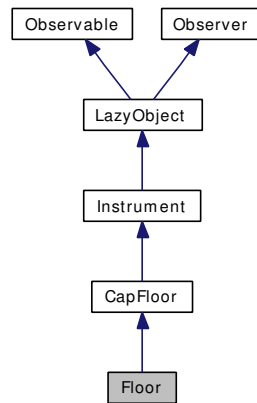
#### Public Member Functions

- **FloatingTypePayoff** (Option::Type type)
- Real **operator()** (Real price) const
- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.383 Floor Class Reference

```
#include <ql/Instruments/capfloor.hpp>
```

Inheritance diagram for Floor:



### 7.383.1 Detailed Description

Concrete floor class.

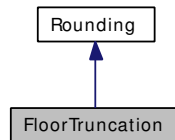
#### Public Member Functions

- **Floor** (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &floatingLeg, const std::vector< [Rate](#) > &exerciseRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared\_ptr< [PricingEngine](#) > &engine)

## 7.384 FloorTruncation Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for FloorTruncation:



### 7.384.1 Detailed Description

[Floor](#) truncation.

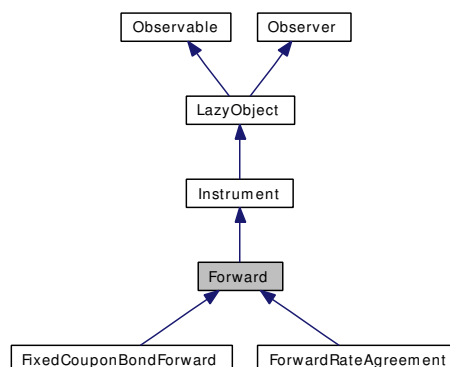
#### Public Member Functions

- `FloorTruncation` ([Integer](#) precision, [Integer](#) digit=5)

## 7.385 Forward Class Reference

```
#include <ql/Instruments/forward.hpp>
```

Inheritance diagram for Forward:



### 7.385.1 Detailed Description

Abstract base forward class.

Derived classes must implement the virtual functions `spotValue()` (NPV or spot price) and `spotIncome()` associated with the specific relevant underlying (e.g. bond, stock, commodity, loan/deposit). These functions must be used to set the protected member variables `underlyingSpotValue_` and `underlyingIncome_` within `performCalculations()` in the derived class before the base-class implementation is called.

`spotIncome()` refers generically to the present value of coupons, dividends or storage costs.

`discountCurve_` is the curve used to discount forward contract cash flows back to the evaluation day, as well as to obtain forward values for spot values/prices.

`incomeDiscountCurve_`, which for generality is not automatically set to the `discountCurve_`, is the curve used to discount future income/dividends/storage-costs etc back to the evaluation date.

#### Todo

Add preconditions and tests

#### Warning

This class still needs to be rigorously tested

### Public Member Functions

- virtual Real `spotValue()` const=0  
*returns spot value/price of an underlying financial instrument*
- virtual Real `spotIncome` (const `Handle< YieldTermStructure >` &incomeDiscountCurve) const=0  
*NPV of income/dividends/storage-costs etc. of underlying instrument.*

## Inspectors

- virtual [Date](#) **settlementDate** () const
- const [Calendar](#) & **calendar** () const
- [BusinessDayConvention](#) **businessDayConvention** () const
- const [DayCounter](#) & **dayCounter** () const
- boost::shared\_ptr< [YieldTermStructure](#) > **discountCurve** () const  
*term structure relevant to the contract (e.g. repo curve)*
- boost::shared\_ptr< [YieldTermStructure](#) > **incomeDiscountCurve** () const  
*term structure that discounts the underlying's income cash flows*
- bool **isExpired** () const  
*returns whether the instrument is still tradable.*

## Calculations

- virtual Real **forwardValue** () const  
*forward value/price of underlying, discounting income/dividends*
- [InterestRate](#) **impliedYield** (Real underlyingSpotValue, Real forwardValue, [Date](#) settlementDate, Compounding compoundingConvention, [DayCounter](#) dayCount)

## Protected Member Functions

- **Forward** (const [DayCounter](#) &dayCount, const [Calendar](#) &calendar, [BusinessDayConvention](#) businessDayConvention, Integer settlementDays, const boost::shared\_ptr< [Payoff](#) > &payoff, const [Date](#) &valueDate, const [Date](#) &maturityDate, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())
- void **performCalculations** () const

## Protected Attributes

- Real **underlyingIncome\_**
- Real **underlyingSpotValue\_**
- [DayCounter](#) **dayCount\_**
- [Calendar](#) **calendar\_**
- [BusinessDayConvention](#) **businessDayConvention\_**
- Integer **settlementDays\_**
- boost::shared\_ptr< [Payoff](#) > **payoff\_**
- [Date](#) **valueDate\_**
- [Date](#) **maturityDate\_**  
*maturityDate of the forward contract or delivery date of underlying*
- [Handle](#)< [YieldTermStructure](#) > **discountCurve\_**
- [Handle](#)< [YieldTermStructure](#) > **incomeDiscountCurve\_**

## 7.385.2 Member Function Documentation

### 7.385.2.1 virtual Real forwardValue () const [virtual]

forward value/price of underlying, discounting income/dividends

**Note:**

if this is a bond forward price, it must be a dirty forward price.

### 7.385.2.2 InterestRate impliedYield (Real underlyingSpotValue, Real forwardValue, Date settlementDate, Compounding compoundingConvention, DayCounter dayCount)

Simple yield calculation based on underlying spot and forward values, taking into account underlying income. When  $t > 0$ , call with: `underlyingSpotValue=spotValue(t)`, `forwardValue=strikePrice`, to get current yield. For a repo, if  $t = 0$ , `impliedYield` should reproduce the spot repo rate. For FRA's, this should reproduce the relevant zero rate at the FRA's `maturityDate_`;

### 7.385.2.3 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](#).

Reimplemented in [FixedCouponBondForward](#), and [ForwardRateAgreement](#).

## 7.385.3 Member Data Documentation

### 7.385.3.1 Real underlyingIncome\_ [mutable, protected]

derived classes must set this, typically via [spotIncome\(\)](#)

### 7.385.3.2 Real underlyingSpotValue\_ [mutable, protected]

derived classes must set this, typically via [spotValue\(\)](#)

### 7.385.3.3 Date valueDate\_ [protected]

`valueDate` = settlement date (date the fwd contract starts accruing)

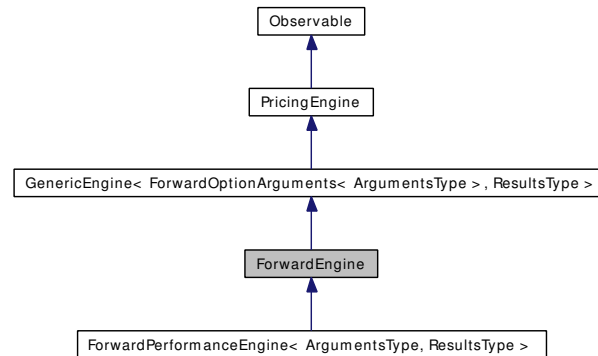
### 7.385.3.4 Handle<YieldTermStructure> incomeDiscountCurve\_ [protected]

must set this in derived classes, based on particular underlying

## 7.386 ForwardEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

Inheritance diagram for ForwardEngine:



### 7.386.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::ForwardEngine<
ArgumentsType, ResultsType >
```

[Forward](#) engine base class.

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

### Public Member Functions

- **ForwardEngine** (const boost::shared\_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > &)
- void **setOriginalArguments** () const
- void **calculate** () const
- void **getOriginalResults** () const

### Protected Attributes

- boost::shared\_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > > **originalEngine\_**
- ArgumentsType \* **originalArguments\_**
- const ResultsType \* **originalResults\_**



## 7.387 ForwardFlat Class Reference

```
#include <ql/Math/forwardflatinterpolation.hpp>
```

### 7.387.1 Detailed Description

Forward-flat interpolation factory and traits.

#### Public Types

- enum { **global** = 0 }

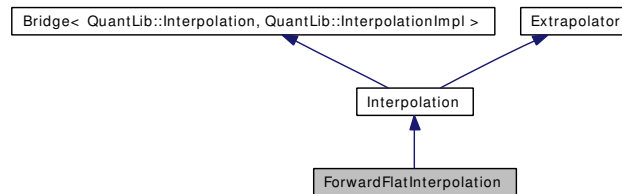
#### Public Member Functions

- template<class I1, class I2> [Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

## 7.388 ForwardFlatInterpolation Class Reference

```
#include <ql/Math/forwardflatinterpolation.hpp>
```

Inheritance diagram for ForwardFlatInterpolation:



### 7.388.1 Detailed Description

Forward-flat interpolation between discrete points.

#### Public Member Functions

- `template<class I1, class I2> ForwardFlatInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)`

### 7.388.2 Constructor & Destructor Documentation

#### 7.388.2.1 [ForwardFlatInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

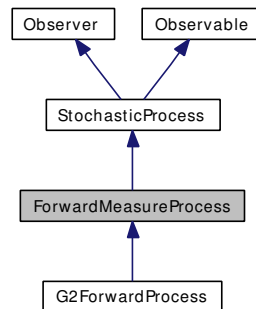
**Precondition:**

the  $x$  values must be sorted.

## 7.389 ForwardMeasureProcess Class Reference

```
#include <ql/Processes/forwardmeasureprocess.hpp>
```

Inheritance diagram for ForwardMeasureProcess:



### 7.389.1 Detailed Description

forward-measure stochastic process

stochastic process whose dynamics are expressed in the forward measure.

### Public Member Functions

- void **setForwardMeasureTime** (Time)

### Protected Member Functions

- **ForwardMeasureProcess** (Time T)
- **ForwardMeasureProcess** (const boost::shared\_ptr< discretization > &)

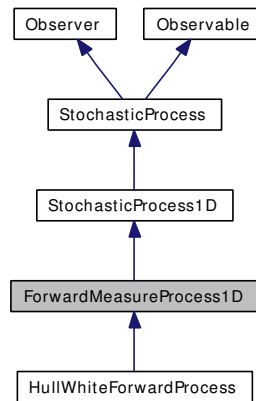
### Protected Attributes

- Time T\_

## 7.390 ForwardMeasureProcess1D Class Reference

#include <ql/Processes/forwardmeasureprocess.hpp>

Inheritance diagram for ForwardMeasureProcess1D:



### 7.390.1 Detailed Description

forward-measure 1-D stochastic process

1-D stochastic process whose dynamics are expressed in the forward measure.

#### Public Member Functions

- void **setForwardMeasureTime** (Time)

#### Protected Member Functions

- **ForwardMeasureProcess1D** (Time T)
- **ForwardMeasureProcess1D** (const boost::shared\_ptr< discretization > &)

#### Protected Attributes

- Time T\_

## 7.391 ForwardOptionArguments Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

### 7.391.1 Detailed Description

```
template<class ArgumentsType> class QuantLib::ForwardOptionArguments< Arguments-  
Type >
```

Arguments for forward (strike-resetting) option calculation

### Public Member Functions

- void `validate ()` const

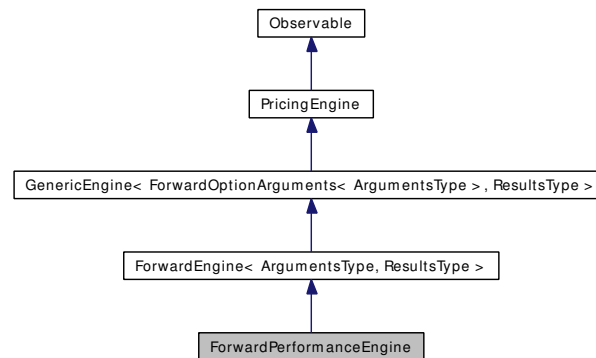
### Public Attributes

- Real `moneyness`
- [Date](#) `resetDate`

## 7.392 ForwardPerformanceEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardperformanceengine.hpp>
```

Inheritance diagram for ForwardPerformanceEngine:



### 7.392.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::ForwardPerformance-
Engine< ArgumentsType, ResultsType >
```

[Forward](#) performance engine.

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

### Public Member Functions

- **ForwardPerformanceEngine** (const boost::shared\_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > > &)
- void **calculate** () const
- void **getOriginalResults** () const

## 7.393 ForwardRate Struct Reference

```
#include <ql/TermStructures/bootstraptraits.hpp>
```

### 7.393.1 Detailed Description

Forward-curve traits.

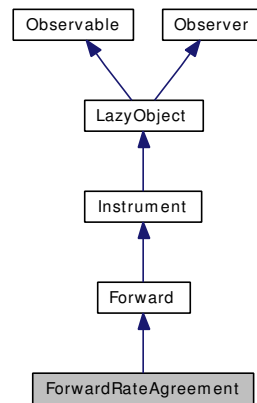
#### Static Public Member Functions

- static [Rate](#) **initialValue** ()
- static [Rate](#) **initialGuess** ()
- static [Rate](#) **guess** (const [YieldTermStructure](#) \*c, const [Date](#) &d)
- static [Rate](#) **minValueAfter** (Size, const std::vector< [Real](#) > &)
- static [Rate](#) **maxValueAfter** (Size, const std::vector< [Real](#) > &)
- static void **updateGuess** (std::vector< [Rate](#) > &data, [Rate](#) forward, Size i)

## 7.394 ForwardRateAgreement Class Reference

```
#include <ql/Instruments/forwardrateagreement.hpp>
```

Inheritance diagram for ForwardRateAgreement:



### 7.394.1 Detailed Description

[Forward](#) rate agreement (FRA) class.

1. Unlike the forward contract conventions on carryable financial assets (stocks, bonds, commodities), the valueDate for an FRA is taken to be the day the forward loan or deposit begins and when full settlement takes place (based on the NPV of the contract on that date). maturityDate is the date the forward loan or deposit ends. In fact, the FRA settles and expires on the valueDate, not on the (later) maturityDate. It follows that (maturityDate - valueDate) is the tenor/term of the underlying loan or deposit
2. Choose position type = Long for an "FRA purchase" (future long loan, short deposit [borrower])
3. Choose position type = [Short](#) for an "FRA sale" (future short loan, long deposit [lender])
4. If strike is given in the constructor, can calculate the NPV of the contract via [NPV\(\)](#).
5. If forward rate is desired/unknown, it can be obtained via [forwardRate\(\)](#). In this case, the strike variable in the constructor is irrelevant and will be ignored.

**Example:** [valuation of a forward-rate agreement](#)

#### [Todo](#)

Add preconditions and tests

#### [Todo](#)

Should put an instance of [ForwardRateAgreement](#) in the [FraRateHelper](#) to ensure consistency with the piecewise yield curve.

#### [Todo](#)

Differentiate between BBA (British)/AFB (French) [assumed here] and ABA (Australian) banker conventions in the calculations.



### Warning

This class still needs to be rigorously tested

### Examples:

[FRA.cpp](#).

## Public Member Functions

- **ForwardRateAgreement** (const [Date](#) &valueDate, const [Date](#) &maturityDate, Position::Type type, [Rate](#) strikeForwardRate, Real notionalAmount, const boost::shared\_ptr<[Xibor](#) > &index, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())

### Calculations

- bool [isExpired](#) () const
- [Date](#) [settlementDate](#) () const
- Real [spotIncome](#) (const [Handle](#)< [YieldTermStructure](#) > &incomeDiscountCurve) const
- Real [spotValue](#) () const  
*Spot value (NPV) of the underlying loan.*
- [InterestRate](#) [forwardRate](#) () const  
*Returns the relevant forward rate associated with the FRA term.*

## Protected Member Functions

- void [performCalculations](#) () const

## Protected Attributes

- Position::Type [fraType\\_](#)
- [InterestRate](#) [forwardRate\\_](#)  
*aka FRA rate (the market forward rate)*
- [InterestRate](#) [strikeForwardRate\\_](#)  
*aka FRA fixing rate, contract rate*
- Real [notionalAmount\\_](#)
- boost::shared\_ptr<[Xibor](#) > [index\\_](#)

## 7.394.2 Member Function Documentation

### 7.394.2.1 bool isExpired () const [virtual]

A FRA expires/settles on the valueDate

Reimplemented from [Forward](#).

**7.394.2.2   [Date](#) `settlementDate () const`   [virtual]**

This returns `evaluationDate + settlementDays` (not FRA `valueDate`).

Reimplemented from [Forward](#).

**7.394.2.3   `Real spotIncome (const Handle< YieldTermStructure > & incomeDiscountCurve) const`   [virtual]**

Income is zero for a FRA

Implements [Forward](#).

**7.394.2.4   `Real spotValue () const`   [virtual]**

Spot value (NPV) of the underlying loan.

This has always a positive value (asset), even if short the FRA

Implements [Forward](#).

**7.394.2.5   `void performCalculations () const`   [protected, virtual]**

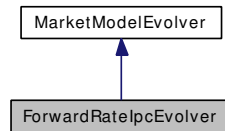
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Forward](#).

## 7.395 ForwardRateIpcEvolver Class Reference

```
#include <ql/MarketModels/Evolvers/forwardrateipcevolver.hpp>
```

Inheritance diagram for ForwardRateIpcEvolver:



### 7.395.1 Detailed Description

Iterative Predictor-Corrector.

#### Public Member Functions

- **ForwardRateIpcEvolver** (const boost::shared\_ptr< MarketModel > &, const Brownian-GeneratorFactory &, const std::vector< Size > &numeraires)

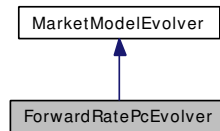
#### MarketModelEvolver interface

- const std::vector< Size > & **numeraires** () const
- Real **startNewPath** ()
- Real **advanceStep** ()
- Size **currentStep** () const
- const [CurveState](#) & **currentState** () const

## 7.396 ForwardRatePcEvolver Class Reference

```
#include <ql/MarketModels/Evolvers/forwardratepcevolver.hpp>
```

Inheritance diagram for ForwardRatePcEvolver:



### 7.396.1 Detailed Description

Predictor-Corrector.

#### Public Member Functions

- **ForwardRatePcEvolver** (const boost::shared\_ptr< MarketModel > &, const Brownian-GeneratorFactory &, const std::vector< Size > &numeraires)

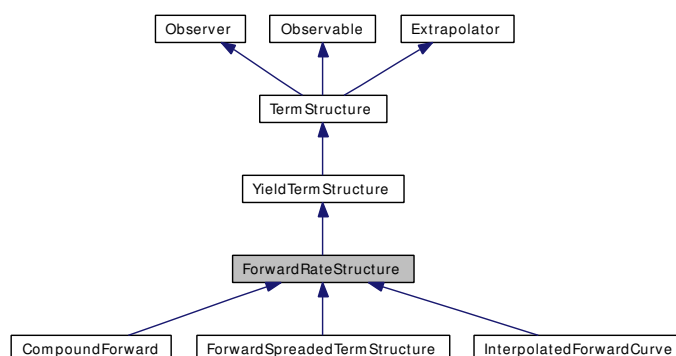
#### MarketModelEvolver interface

- const std::vector< Size > & **numeraires** () const
- Real **startNewPath** ()
- Real **advanceStep** ()
- Size **currentStep** () const
- const [CurveState](#) & **currentState** () const

## 7.397 ForwardRateStructure Class Reference

#include <ql/TermStructures/forwardstructure.hpp>

Inheritance diagram for ForwardRateStructure:



### 7.397.1 Detailed Description

**Forward** rate term structure.

This abstract class acts as an adapter to [TermStructure](#) allowing the programmer to implement only the `forwardImpl(const Date&, bool)` method in derived classes. Zero yields and discounts are calculated from forwards.

Rates are assumed to be annual continuous compounding.

### Protected Member Functions

#### YieldTermStructure implementation

- [DiscountFactor](#) `discountImpl (Time) const`
- virtual [Rate](#) `forwardImpl (Time) const=0`  
*instantaneous forward-rate calculation*
- virtual [Rate](#) `zeroYieldImpl (Time) const`

### 7.397.2 Member Function Documentation

#### 7.397.2.1 [DiscountFactor](#) `discountImpl (Time) const` [protected, virtual]

Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Implements [YieldTermStructure](#).

Reimplemented in [CompoundForward](#).

#### 7.397.2.2 [Rate](#) `zeroYieldImpl (Time) const` [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

**Warning**

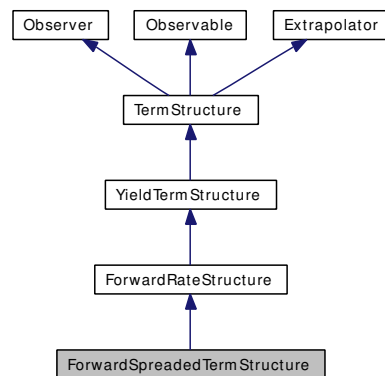
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own `zeroYield` method.

Reimplemented in [CompoundForward](#), [InterpolatedForwardCurve](#), and [ForwardSpreaded-TermStructure](#).

## 7.398 ForwardSpreadedTermStructure Class Reference

```
#include <ql/TermStructures/forwardspreadedtermstructure.hpp>
```

Inheritance diagram for ForwardSpreadedTermStructure:



### 7.398.1 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.

#### Tests

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

### Public Member Functions

- **ForwardSpreadedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Handle](#)< [Quote](#) > &spread)

#### YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Calendar](#) [calendar](#) () const  
*the calendar used for reference date calculation*
- const [Date](#) & [referenceDate](#) () const  
*the date at which discount = 1.0 and/or variance = 0.0*

- [Date maxDate](#) () const  
*the latest date for which the curve can return values*
- [Time maxTime](#) () const  
*the latest time for which the curve can return values*

### Protected Member Functions

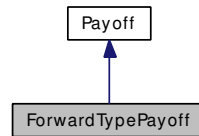
- [Rate forwardImpl](#) (Time) const  
*returns the spreaded forward rate*
- [Rate zeroYieldImpl](#) (Time) const  
*returns the spreaded zero yield rate*



## 7.399 ForwardTypePayoff Class Reference

```
#include <ql/Instruments/forward.hpp>
```

Inheritance diagram for ForwardTypePayoff:



### 7.399.1 Detailed Description

Class for forward type payoffs.

#### Public Member Functions

- **ForwardTypePayoff** (Position::Type type, Real strike)
- Position::Type **forwardType** () const
- Real **strike** () const
- Real **operator()** (Real price) const

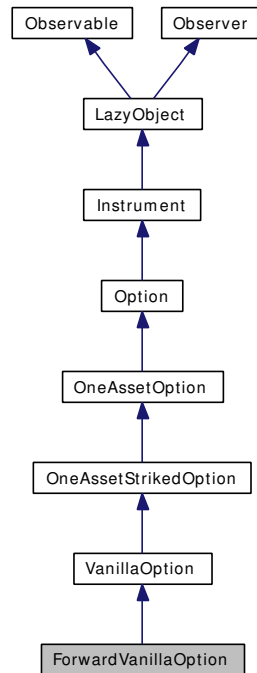
#### Protected Attributes

- Position::Type **type\_**
- Real **strike\_**

## 7.400 ForwardVanillaOption Class Reference

```
#include <ql/Instruments/forwardvanillaoption.hpp>
```

Inheritance diagram for ForwardVanillaOption:



### 7.400.1 Detailed Description

[Forward](#) version of a vanilla option.

#### Public Types

- typedef [ForwardOptionArguments](#)< [VanillaOption::arguments](#) > **arguments**
- typedef [VanillaOption::results](#) **results**
- typedef [ForwardEngine](#)< [VanillaOption::arguments](#), [VanillaOption::results](#) > **engine**

#### Public Member Functions

- **ForwardVanillaOption** (Real moneyness, [Date](#) resetDate, const boost::shared\_ptr< [StochasticProcess](#) > &stochProc, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine)
- void [setupArguments](#) ([Arguments](#) \*) const
- void [fetchResults](#) (const [Results](#) \*) const

## 7.400.2 Member Function Documentation

### 7.400.2.1 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 [OneAssetStrikedOption](#).

### 7.400.2.2 void fetchResults (const [Results](#) \*) const [virtual]

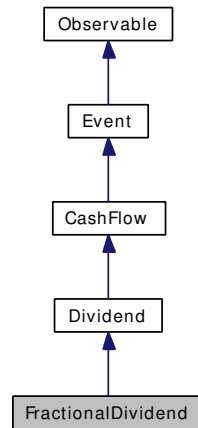
When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetStrikedOption](#).

## 7.401 FractionalDividend Class Reference

```
#include <ql/CashFlows/dividend.hpp>
```

Inheritance diagram for FractionalDividend:



### 7.401.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

#### Public Member Functions

- **FractionalDividend** (Real rate, const [Date](#) &date)
- **FractionalDividend** (Real rate, Real nominal, const [Date](#) &date)

#### Dividend interface

- virtual Real [amount](#) () const  
*returns the amount of the cash flow*
- virtual Real **amount** (Real underlying) const

#### Inspectors

- Real **rate** () const
- Real **nominal** () const

#### Protected Attributes

- Real **rate\_**
- Real **nominal\_**

## 7.401.2 Member Function Documentation

### 7.401.2.1 virtual Real amount () const [virtual]

returns the amount of the cash flow

**Note:**

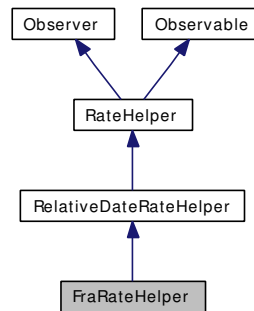
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [Dividend](#).

## 7.402 FraRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for FraRateHelper:



### 7.402.1 Detailed Description

Rate helper for bootstrapping over FRA rates.

Examples:

[FRA.cpp](#), and [swapvaluation.cpp](#).

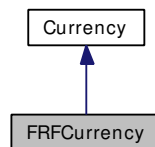
### Public Member Functions

- **FraRateHelper** (const [Handle](#)< [Quote](#) > &rate, Integer monthsToStart, Integer monthsToEnd, Integer settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **FraRateHelper** ([Rate](#) rate, Integer monthsToStart, Integer monthsToEnd, Integer settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **Real impliedQuote** () const
- **DiscountFactor discountGuess** () const

## 7.403 FRFCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for FRFCurrency:



### 7.403.1 Detailed Description

French franc.

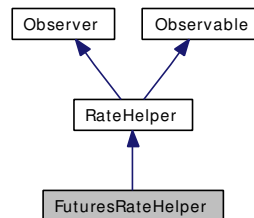
The ISO three-letter code was FRF; the numeric code was 250. It was divided in 100 centimes.

Obsoleted by the Euro since 1999.

## 7.404 FuturesRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for FuturesRateHelper:



### 7.404.1 Detailed Description

Rate helper for bootstrapping over interest-rate futures prices.

#### Todo

implement/refactor constructors with: [Index](#) instead of (nMonths, calendar, convention, dayCounter), [IMM](#) code

#### Examples:

[swapvaluation.cpp](#).

### Public Member Functions

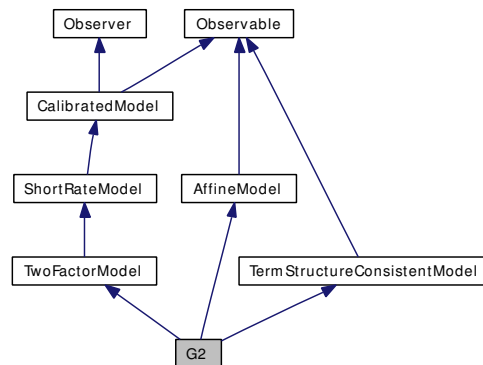
- **FuturesRateHelper** (const [Handle](#)< [Quote](#) > &price, const [Date](#) &immDate, [Integer](#) nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, const [Handle](#)< [Quote](#) > &convexityAdjustment)
- **FuturesRateHelper** (const [Handle](#)< [Quote](#) > &price, const [Date](#) &immDate, [Integer](#) nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, [Rate](#) convexityAdjustment=0.0)
- **FuturesRateHelper** ([Real](#) price, const [Date](#) &immDate, [Integer](#) nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, [Rate](#) convexityAdjustment=0.0)
- **[Real](#) impliedQuote** () const
- **[DiscountFactor](#) discountGuess** () const
- **[Real](#) convexityAdjustment** () const



## 7.405 G2 Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Inheritance diagram for G2:



### 7.405.1 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  $dW_t^1 dW_t^2 = \rho dt$ .

#### Bug

This class was not tested enough to guarantee its functionality.

#### Examples:

[BermudanSwaption.cpp](#).

### Public Member Functions

- **G2** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, Real a=0.1, Real sigma=0.01, Real b=0.1, Real eta=0.01, Real rho=-0.75)
- **boost::shared\_ptr**< [ShortRateDynamics](#) > **dynamics** () const  
*Returns the short-rate dynamics.*
- virtual Real **discountBond** (Time now, Time maturity, [Array](#) factors) const
- Real **discountBond** (Time, Time, [Rate](#), [Rate](#)) const

- Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bond-Maturity) const
- Real **swaption** (const [Swaption::arguments](#) &arguments, Real range, Size intervals) const
- [DiscountFactor](#) **discount** (Time t) const

*Implied discount curve.*

## Protected Member Functions

- void **generateArguments** ()
- Real **A** (Time t, Time T) const
- Real **B** (Real x, Time t) const

## Friends

- class **SwaptionPricingFunction**

## Classes

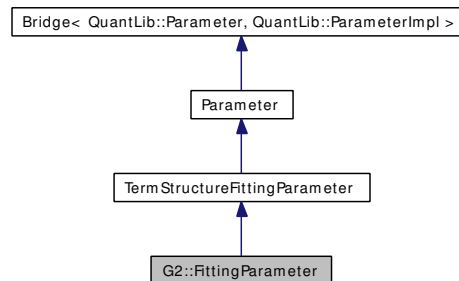
- class [FittingParameter](#)

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

## 7.406 G2::FittingParameter Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Inheritance diagram for G2::FittingParameter:



### 7.406.1 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  $t$ .

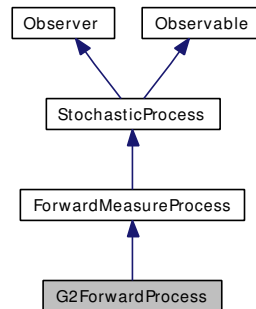
### Public Member Functions

- **FittingParameter** (const [Handle< YieldTermStructure >](#) &termStructure, [Real](#) a, [Real](#) sigma, [Real](#) b, [Real](#) eta, [Real](#) rho)

## 7.407 G2ForwardProcess Class Reference

```
#include <ql/Processes/g2process.hpp>
```

Inheritance diagram for G2ForwardProcess:



### 7.407.1 Detailed Description

forward G2 stochastic process

#### Public Member Functions

- **G2ForwardProcess** (Real a, Real sigma, Real b, Real eta, Real rho)

#### StochasticProcess interface

- **Size** [size](#) () const  
*returns the number of dimensions of the stochastic process*
- **Disposable**< [Array](#) > **initialValues** () const  
*returns the initial values of the state variables*
- **Disposable**< [Array](#) > **drift** (Time t, const [Array](#) &x) const  
*returns the drift part of the equation, i.e.,  $\mu(t, x_t)$*
- **Disposable**< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*
- **Disposable**< [Array](#) > **expectation** (Time t0, const [Array](#) &x0, Time dt) const
- **Disposable**< [Matrix](#) > **stdDeviation** (Time t0, const [Array](#) &x0, Time dt) const
- **Disposable**< [Matrix](#) > **covariance** (Time t0, const [Array](#) &x0, Time dt) const

#### Protected Member Functions

- Real **xForwardDrift** (Time t, Time T) const
- Real **yForwardDrift** (Time t, Time T) const
- Real **Mx\_T** (Real s, Real t, Real T) const
- Real **My\_T** (Real s, Real t, Real T) const

## Protected Attributes

- Real `x0_`
- Real `y0_`
- Real `a_`
- Real `sigma_`
- Real `b_`
- Real `eta_`
- Real `rho_`
- `boost::shared_ptr< QuantLib::OrnsteinUhlenbeckProcess > xProcess_`
- `boost::shared_ptr< QuantLib::OrnsteinUhlenbeckProcess > yProcess_`

## 7.407.2 Member Function Documentation

### 7.407.2.1 `Disposable<Array> expectation (Time t0, const Array & x0, Time dt) const` [virtual]

returns the expectation  $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

### 7.407.2.2 `Disposable<Matrix> stdDeviation (Time t0, const Array & x0, Time dt) const` [virtual]

returns the standard deviation  $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

### 7.407.2.3 `Disposable<Matrix> covariance (Time t0, const Array & x0, Time dt) const` [virtual]

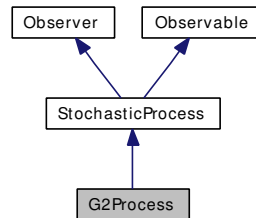
returns the covariance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

## 7.408 G2Process Class Reference

```
#include <ql/Processes/g2process.hpp>
```

Inheritance diagram for G2Process:



### 7.408.1 Detailed Description

G2 stochastic process

#### Public Member Functions

- **G2Process** (Real a, Real sigma, Real b, Real eta, Real rho)
- Real **x0** () const
- Real **y0** () const
- Real **a** () const
- Real **sigma** () const
- Real **b** () const
- Real **eta** () const
- Real **rho** () const

#### StochasticProcess interface

- Size **size** () const  
*returns the number of dimensions of the stochastic process*
- Disposable< Array > **initialValues** () const  
*returns the initial values of the state variables*
- Disposable< Array > **drift** (Time t, const Array &x) const  
*returns the drift part of the equation, i.e.,  $\mu(t, x_t)$*
- Disposable< Matrix > **diffusion** (Time t, const Array &x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*
- Disposable< Array > **expectation** (Time t0, const Array &x0, Time dt) const
- Disposable< Matrix > **stdDeviation** (Time t0, const Array &x0, Time dt) const
- Disposable< Matrix > **covariance** (Time t0, const Array &x0, Time dt) const

## 7.408.2 Member Function Documentation

### 7.408.2.1 **Disposable**<**Array**> expectation (Time $t_0$ , const **Array** & $x_0$ , Time $dt$ ) const [virtual]

returns the expectation  $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

### 7.408.2.2 **Disposable**<**Matrix**> stdDeviation (Time $t_0$ , const **Array** & $x_0$ , Time $dt$ ) const [virtual]

returns the standard deviation  $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

### 7.408.2.3 **Disposable**<**Matrix**> covariance (Time $t_0$ , const **Array** & $x_0$ , Time $dt$ ) const [virtual]

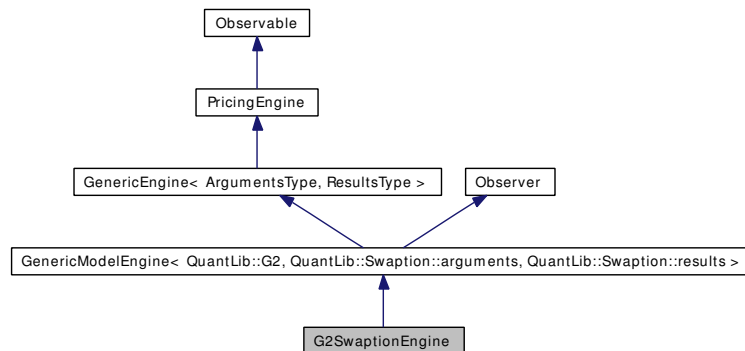
returns the covariance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

## 7.409 G2SwaptionEngine Class Reference

```
#include <ql/PricingEngines/Swaption/g2swaptionengine.hpp>
```

Inheritance diagram for G2SwaptionEngine:



### 7.409.1 Detailed Description

Swaption priced by means of the Black formula

#### Warning

The engine assumes that the exercise date equals the start date of the passed swap.

#### Examples:

[BermudanSwaption.cpp](#).

### Public Member Functions

- **G2SwaptionEngine** (const boost::shared\_ptr< [G2](#) > &mod, Real range, Size intervals)
- void **calculate** () const



## 7.410 GammaFunction Class Reference

```
#include <ql/Math/gammadistribution.hpp>
```

### 7.410.1 Detailed Description

Gamma function class.

This is a function defined by

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt$$

The implementation of the algorithm was inspired by "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery, chapter 6

#### Tests

the correctness of the returned value is tested by checking it against known good results.

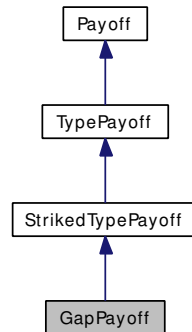
### Public Member Functions

- Real **logValue** (Real x) const

## 7.411 GapPayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for GapPayoff:



### 7.411.1 Detailed Description

Binary gap payoff.

#### Public Member Functions

- **GapPayoff** (Option::Type type, Real strike, Real strikePayoff)
- Real **operator()** (Real price) const
- Real **strikePayoff** () const
- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.412 Garch11 Class Reference

```
#include <ql/VolatilityModels/garch.hpp>
```

### 7.412.1 Detailed Description

GARCH volatility model.

Volatilities are assumed to be expressed on an annual basis.

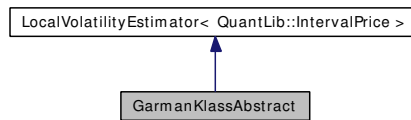
### Public Member Functions

- **Garch11** (Real a, Real b, Real vl)
- **Garch11** (const [TimeSeries](#)< Volatility > &qs)
- [TimeSeries](#)< Volatility > **calculate** (const [TimeSeries](#)< Volatility > &quoteSeries)
- [TimeSeries](#)< Volatility > **calculate** (const [TimeSeries](#)< Volatility > &quoteSeries, Real, Real, Real)
- void **calibrate** (const [TimeSeries](#)< Volatility > &quoteSeries)

## 7.413 GarmanKlassAbstract Class Reference

```
#include <ql/VolatilityModels/garmanklass.hpp>
```

Inheritance diagram for GarmanKlassAbstract:



### 7.413.1 Detailed Description

This class implements a concrete volatility model based on high low formulas using the method of Garman and Klass in their paper "On the Estimation of the Security Price from Historical Data" at [http://www.fea.com/resources/pdf/a\\_estimation\\_of\\_security\\_price.pdf](http://www.fea.com/resources/pdf/a_estimation_of_security_price.pdf)

Volatilities are assumed to be expressed on an annual basis.

#### Public Member Functions

- **GarmanKlassAbstract** (Real y)
- **TimeSeries< Volatility > calculate** (const **TimeSeries< IntervalPrice >** &quoteSeries)

#### Protected Member Functions

- virtual Real **calculatePoint** (const **IntervalPrice** &p)=0

#### Protected Attributes

- Real **yearFraction\_**

## 7.414 GarmanKlassOpenClose Class Template Reference

```
#include <ql/VolatilityModels/garmanklass.hpp>
```

### 7.414.1 Detailed Description

```
template<class T> class QuantLib::GarmanKlassOpenClose< T >
```

This template factors out common functionality found in classes which rely on the difference between the previous day's close price and today's open price.

### Public Member Functions

- `GarmanKlassOpenClose` (Real y, Real marketOpenFraction, Real a)
- `TimeSeries< Volatility > calculate` (const `TimeSeries< IntervalPrice >` &quoteSeries)

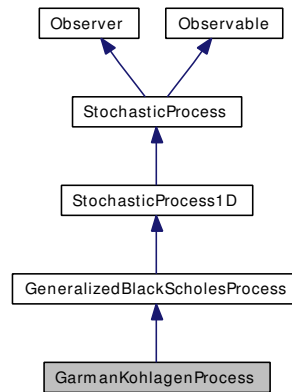
### Protected Attributes

- Real `f_`
- Real `a_`

## 7.415 GarmanKohlagenProcess Class Reference

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Inheritance diagram for GarmanKohlagenProcess:



### 7.415.1 Detailed Description

Garman-Kohlhagen (1983) stochastic process.

This class describes the stochastic process for an exchange rate given by

$$dS(t, S) = (r(t) - r_f(t) - \frac{\sigma(t, S)^2}{2})dt + \sigma dW_t.$$

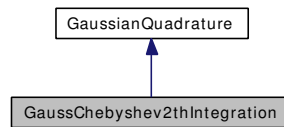
### Public Member Functions

- **GarmanKohlagenProcess** (const [Handle](#)< [Quote](#) > &x0, const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [YieldTermStructure](#) > &domesticRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &blackVolTS, const boost::shared\_ptr< discretization > &d=boost::shared\_ptr< discretization >(new [EulerDiscretization](#)))

## 7.416 GaussChebyshev2thIntegration Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

Inheritance diagram for GaussChebyshev2thIntegration:



### 7.416.1 Detailed Description

Gauss-Chebyshev integration second kind.

This class performs a 1-dimensional Gauss-Chebyshev integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{1/2}$$

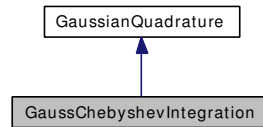
### Public Member Functions

- **GaussChebyshev2thIntegration** (Size n)

## 7.417 GaussChebyshevIntegration Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

Inheritance diagram for GaussChebyshevIntegration:



### 7.417.1 Detailed Description

Gauss-Chebyshev integration.

This class performs a 1-dimensional Gauss-Chebyshev integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{-1/2}$$

### Public Member Functions

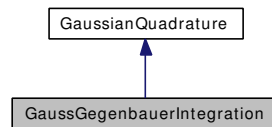
- `GaussChebyshevIntegration` (Size n)



## 7.418 GaussGegenbauerIntegration Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

Inheritance diagram for GaussGegenbauerIntegration:



### 7.418.1 Detailed Description

Gauss-Gegenbauer integration.

This class performs a 1-dimensional Gauss-Gegenbauer integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{\lambda-1/2}$$

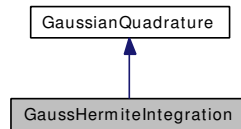
### Public Member Functions

- `GaussGegenbauerIntegration` (Size n, [Real](#) lambda)

## 7.419 GaussHermiteIntegration Class Reference

#include <ql/Math/gaussianquadratures.hpp>

Inheritance diagram for GaussHermiteIntegration:



### 7.419.1 Detailed Description

generalized Gauss-Hermite integration

This class performs a 1-dimensional Gauss-Hermite integration.

$$\int_{-\infty}^{\infty} f(x) dx$$

The weighting function is

$$w(x; \mu) = |x|^{2\mu} \exp -x * x$$

and

$$\mu > -0.5$$

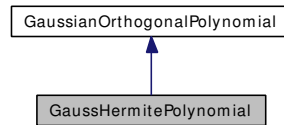
### Public Member Functions

- **GaussHermiteIntegration** (Size n, [Real](#) mu=0.0)

## 7.420 GaussHermitePolynomial Class Reference

```
#include <ql/Math/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussHermitePolynomial:



### 7.420.1 Detailed Description

Gauss-Hermite polynomial.

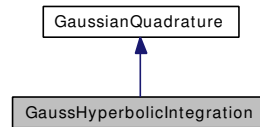
#### Public Member Functions

- **GaussHermitePolynomial** (Real mu=0.0)
- Real **mu\_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

## 7.421 GaussHyperbolicIntegration Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

Inheritance diagram for GaussHyperbolicIntegration:



### 7.421.1 Detailed Description

Gauss-Hyperbolic integration.

This class performs a 1-dimensional Gauss-Hyperbolic integration.

$$\int_{-\infty}^{\infty} f(x) dx$$

The weighting function is

$$w(x) = 1/\cosh(x)$$

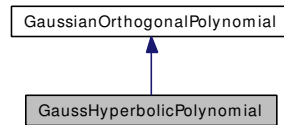
### Public Member Functions

- `GaussHyperbolicIntegration` (Size n)

## 7.422 GaussHyperbolicPolynomial Class Reference

```
#include <ql/Math/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussHyperbolicPolynomial:



### 7.422.1 Detailed Description

Gauss hyperbolic polynomial.

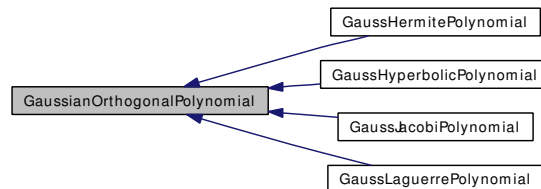
#### Public Member Functions

- Real **mu\_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

## 7.423 GaussianOrthogonalPolynomial Class Reference

```
#include <ql/Math/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussianOrthogonalPolynomial:



### 7.423.1 Detailed Description

orthogonal polynomial for Gaussian quadratures

References: Gauss quadratures and orthogonal polynomials

G.H. Gloub and J.H. Welsch: Calculation of Gauss quadrature rule. Math. Comput. 23 (1986), 221-230

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

The polynomials are defined by the three-term recurrence relation

$$P_{k+1}(x) = (x - \alpha_k)P_k(x) - \beta_k P_{k-1}(x)$$

and

$$\mu_0 = \int w(x)dx$$

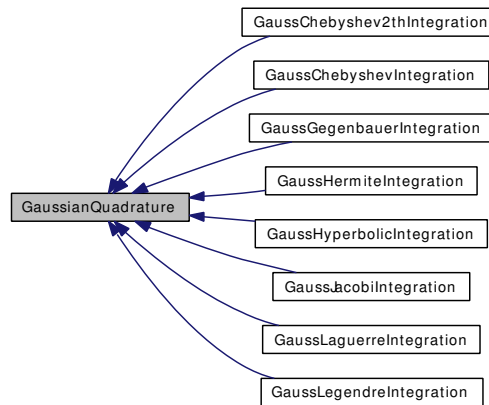
### Public Member Functions

- virtual Real **mu\_0** () const=0
- virtual Real **alpha** (Size i) const=0
- virtual Real **beta** (Size i) const=0
- virtual Real **w** (Real x) const=0
- Real **value** (Size i, Real x) const
- Real **weightedValue** (Size i, Real x) const

## 7.424 GaussianQuadrature Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

Inheritance diagram for GaussianQuadrature:



### 7.424.1 Detailed Description

Integral of a 1-dimensional function using the Gauss quadratures method.

References: Gauss quadratures and orthogonal polynomials

G.H. Gloub and J.H. Welsch: Calculation of Gauss quadrature rule. Math. Comput. 23 (1986), 221-230

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

#### Tests

the correctness of the result is tested by checking it against known good values.

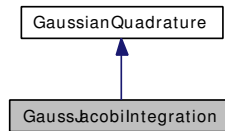
### Public Member Functions

- **GaussianQuadrature** (Size n, const [GaussianOrthogonalPolynomial](#) &p)
- `template<class F> Real operator() (const F &f) const`
- `Size order () const`

## 7.425 GaussJacobiIntegration Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

Inheritance diagram for GaussJacobiIntegration:



### 7.425.1 Detailed Description

Gauss-Jacobi integration.

This class performs a 1-dimensional Gauss-Jacobi integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x; \alpha, \beta) = (1 - x)^\alpha (1 + x)^\beta$$

### Public Member Functions

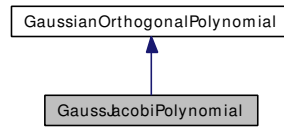
- **GaussJacobiIntegration** (Size n, [Real](#) alpha, [Real](#) beta)



## 7.426 GaussJacobiPolynomial Class Reference

```
#include <ql/Math/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussJacobiPolynomial:



### 7.426.1 Detailed Description

Gauss-Jacobi polynomial.

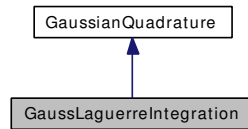
#### Public Member Functions

- **GaussJacobiPolynomial** (Real alpha, Real beta)
- Real **mu\_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

## 7.427 GaussLaguerreIntegration Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

Inheritance diagram for GaussLaguerreIntegration:



### 7.427.1 Detailed Description

generalized Gauss-Laguerre integration

This class performs a 1-dimensional Gauss-Laguerre integration.

$$\int_0^{\infty} f(x) dx$$

The weighting function is

$$w(x; s) = x^s \exp -x$$

and

$$s > -1$$

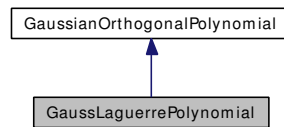
### Public Member Functions

- **GaussLaguerreIntegration** (Size n, [Real](#) s=0.0)

## 7.428 GaussLaguerrePolynomial Class Reference

```
#include <ql/Math/gaussianorthogonalpolynomial.hpp>
```

Inheritance diagram for GaussLaguerrePolynomial:



### 7.428.1 Detailed Description

Gauss-Laguerre polynomial.

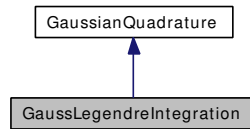
#### Public Member Functions

- **GaussLaguerrePolynomial** (Real s=0.0)
- Real **mu\_0** () const
- Real **alpha** (Size i) const
- Real **beta** (Size i) const
- Real **w** (Real x) const

## 7.429 GaussLegendreIntegration Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

Inheritance diagram for GaussLegendreIntegration:



### 7.429.1 Detailed Description

Gauss-Legendre integration.

This class performs a 1-dimensional Gauss-Legendre integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = 1$$

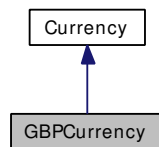
### Public Member Functions

- `GaussLegendreIntegration` (Size n)

## 7.430 GBPCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for GBPCurrency:



### 7.430.1 Detailed Description

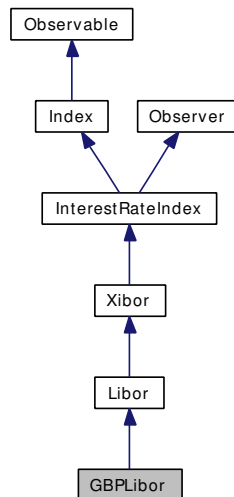
British pound sterling.

The ISO three-letter code is GBP; the numeric code is 826. It is divided into 100 pence.

## 7.431 GBPLibor Class Reference

```
#include <ql/Indexes/gbplibor.hpp>
```

Inheritance diagram for GBPLibor:



### 7.431.1 Detailed Description

GBP LIBOR rate

Pound Sterling LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

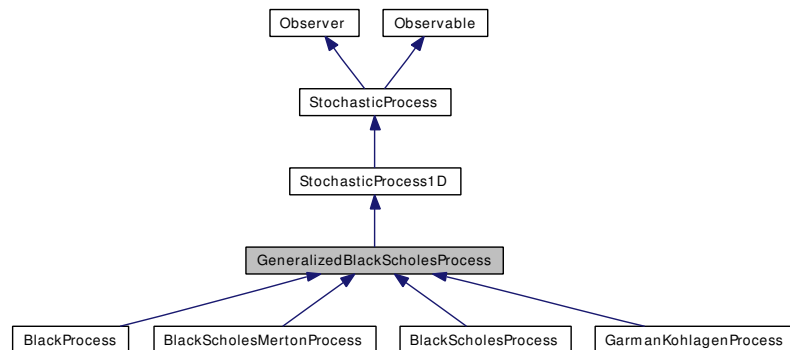
### Public Member Functions

- **GBPLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference)

## 7.432 GeneralizedBlackScholesProcess Class Reference

#include <ql/Processes/blackscholesprocess.hpp>

Inheritance diagram for GeneralizedBlackScholesProcess:



### 7.432.1 Detailed Description

Generalized Black-Scholes stochastic process.

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

- **GeneralizedBlackScholesProcess** (const [Handle< Quote >](#) &x0, const [Handle< YieldTermStructure >](#) &dividendTS, const [Handle< YieldTermStructure >](#) &riskFreeTS, const [Handle< BlackVolTermStructure >](#) &blackVolTS, const boost::shared\_ptr< discretization > &d=boost::shared\_ptr< discretization >(new [EulerDiscretization](#)))
- Time [time](#) (const [Date](#) &) const

#### StochasticProcess1D interface

- Real [x0](#) () const  
*returns the initial value of the state variable*
- Real [drift](#) (Time t, Real x) const
- Real [diffusion](#) (Time t, Real x) const
- Real [apply](#) (Real x0, Real dx) const

#### Observer interface

- void [update](#) ()

#### Inspectors

- const boost::shared\_ptr< [Quote](#) > & [stateVariable](#) () const
- const boost::shared\_ptr< [YieldTermStructure](#) > & [dividendYield](#) () const
- const boost::shared\_ptr< [YieldTermStructure](#) > & [riskFreeRate](#) () const
- const boost::shared\_ptr< [BlackVolTermStructure](#) > & [blackVolatility](#) () const
- const boost::shared\_ptr< [LocalVolTermStructure](#) > & [localVolatility](#) () const

## 7.432.2 Member Function Documentation

### 7.432.2.1 Real drift (Time $t$ , Real $x$ ) const [virtual]

#### Todo

revise extrapolation

Implements [StochasticProcess1D](#).

### 7.432.2.2 Real diffusion (Time $t$ , Real $x$ ) const [virtual]

#### Todo

revise extrapolation

Implements [StochasticProcess1D](#).

### 7.432.2.3 Real apply (Real $x_0$ , Real $dx$ ) const [virtual]

applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented from [StochasticProcess1D](#).

### 7.432.2.4 Time time (const [Date](#) &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

#### Note:

As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

### 7.432.2.5 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 [StochasticProcess](#).



## 7.433 GeneralStatistics Class Reference

```
#include <ql/Math/generalstatistics.hpp>
```

### 7.433.1 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](#). The downside is that it stores all samples, thus increasing the memory requirements.

### Public Types

- typedef Real **value\_type**

### Public Member Functions

#### Inspectors

- Size [samples](#) () const  
*number of samples collected*
- const std::vector< std::pair< Real, Real > > & [data](#) () const  
*collected data*
- Real [weightSum](#) () const  
*sum of data weights*
- Real [mean](#) () const
- Real [variance](#) () const
- Real [standardDeviation](#) () const
- Real [errorEstimate](#) () const
- Real [skewness](#) () const
- Real [kurtosis](#) () const
- Real [min](#) () const
- Real [max](#) () const
- template<class Func, class Predicate> std::pair< Real, Size > [expectationValue](#) (const Func &f, const Predicate &inRange) const
- Real [percentile](#) (Real y) const
- Real [topPercentile](#) (Real y) const

#### Modifiers

- void [add](#) (Real value, Real 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*
- `void sort () const`  
*sort the data set in increasing order*

## 7.433.2 Member Function Documentation

### 7.433.2.1 Real mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

### 7.433.2.2 Real variance () const

returns the variance, defined as

$$\sigma^2 = \frac{N}{N-1} \left\langle (x - \langle x \rangle)^2 \right\rangle.$$

### 7.433.2.3 Real standardDeviation () const

returns the standard deviation  $\sigma$ , defined as the square root of the variance.

### 7.433.2.4 Real errorEstimate () const

returns the error estimate on the mean value, defined as  $\epsilon = \sigma / \sqrt{N}$ .

### 7.433.2.5 Real 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.

### 7.433.2.6 Real 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.

**7.433.2.7 Real min () const**

returns the minimum sample value

**7.433.2.8 Real max () const**

returns the maximum sample value

**7.433.2.9 std::pair<Real,Size> expectationValue (const Func & *f*, const Predicate & *inRange*) const**

Expectation value of a function  $f$  on a given range  $\mathcal{R}$ , i.e.,

$$E[f \mid \mathcal{R}] = \frac{\sum_{x_i \in \mathcal{R}} f(x_i) w_i}{\sum_{x_i \in \mathcal{R}} w_i}.$$

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.

**7.433.2.10 Real percentile (Real *y*) const**

$y$ -th percentile, defined as the value  $\bar{x}$  such that

$$y = \frac{\sum_{x_i < \bar{x}} w_i}{\sum_i w_i}$$

**Precondition:**

$y$  must be in the range  $(0 - 1]$ .

**7.433.2.11 Real topPercentile (Real *y*) const**

$y$ -th top percentile, defined as the value  $\bar{x}$  such that

$$y = \frac{\sum_{x_i > \bar{x}} w_i}{\sum_i w_i}$$

**Precondition:**

$y$  must be in the range  $(0 - 1]$ .

**7.433.2.12 void add (Real *value*, Real *weight* = 1.0)**

adds a datum to the set, possibly with a weight

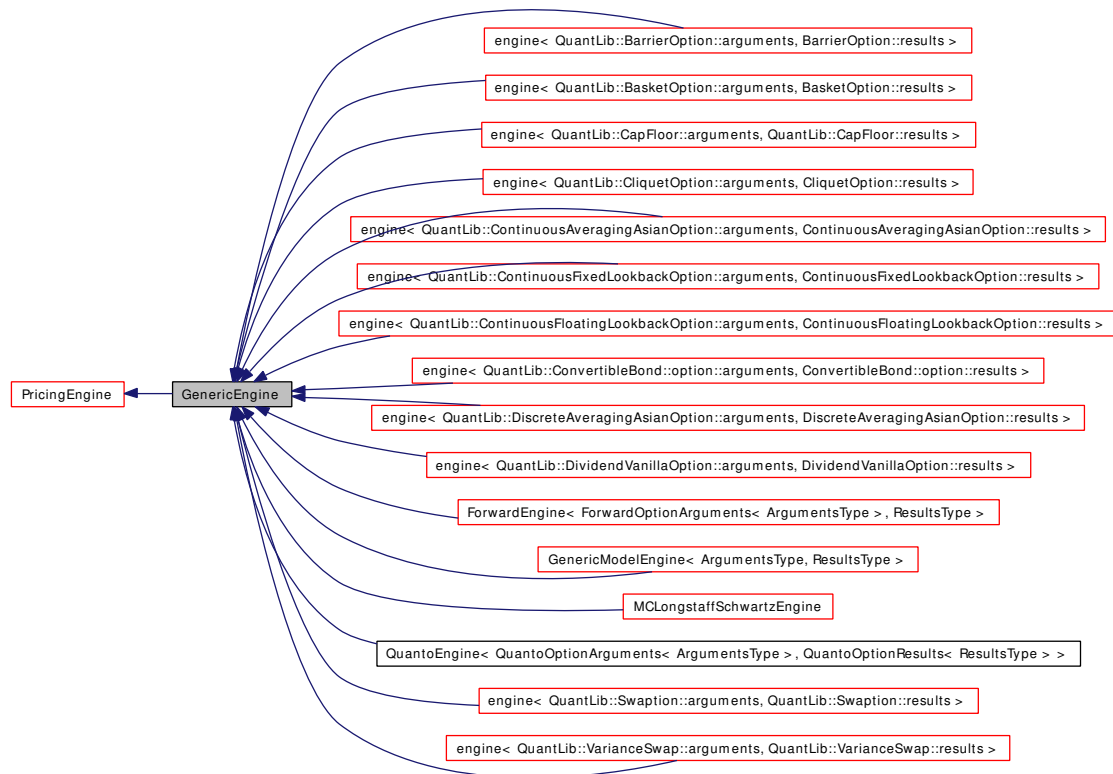
**Precondition:**

weights must be positive or null

## 7.434 GenericEngine Class Template Reference

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

Inheritance diagram for GenericEngine:



### 7.434.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::GenericEngine<
ArgumentsType, ResultsType >
```

template base class for option pricing engines

Derived engines only need to implement the `calculate()` method.

### Public Member Functions

- `Arguments * arguments () const`
- `const Results * results () const`
- `void reset () const`

### Protected Attributes

- `ArgumentsType arguments_`

- ResultsType **results\_**

## 7.435 GenericGaussianStatistics Class Template Reference

```
#include <ql/Math/gaussianstatistics.hpp>
```

### 7.435.1 Detailed Description

```
template<class Stat> class QuantLib::GenericGaussianStatistics< Stat >
```

Statistics tool for gaussian-assumption risk measures.

This class wraps a somewhat generic statistic tool and adds a number of gaussian risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the mean and variance provided by the underlying statistic tool.

### Public Types

- typedef Stat::value\_type **value\_type**

### Public Member Functions

- **GenericGaussianStatistics** (const Stat &s)

#### Gaussian risk measures

- Real [gaussianDownsideVariance](#) () const
- Real [gaussianDownsideDeviation](#) () const
- Real [gaussianRegret](#) (Real target) const
- Real [gaussianPercentile](#) (Real percentile) const
- Real [gaussianTopPercentile](#) (Real percentile) const
- Real [gaussianPotentialUpside](#) (Real percentile) const  
*gaussian-assumption Potential-Upside at a given percentile*
- Real [gaussianValueAtRisk](#) (Real percentile) const  
*gaussian-assumption Value-At-Risk at a given percentile*
- Real [gaussianExpectedShortfall](#) (Real percentile) const  
*gaussian-assumption Expected Shortfall at a given percentile*
- Real [gaussianShortfall](#) (Real target) const  
*gaussian-assumption Shortfall (observations below target)*
- Real [gaussianAverageShortfall](#) (Real target) const  
*gaussian-assumption [Average](#) Shortfall (averaged shortfallness)*

### 7.435.2 Member Function Documentation

#### 7.435.2.1 Real [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  $\theta = 0$  if  $x > 0$  and  $\theta = 1$  if  $x < 0$

#### 7.435.2.2 Real gaussianDownsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

#### 7.435.2.3 Real gaussianRegret (Real *target*) const

returns the variance of observations below target

$$\frac{\sum w_i (\min(0, x_i - \text{target}))^2}{\sum w_i}.$$

See Dembo, Freeman "The Rules Of Risk", Wiley (2001)

#### 7.435.2.4 Real gaussianPercentile (Real *percentile*) const

gaussian-assumption y-th percentile, defined as the value x such that

$$y = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp(-u^2/2) du$$

#### 7.435.2.5 Real gaussianTopPercentile (Real *percentile*) const

**Precondition:**

percentile must be in range (0-100%) extremes excluded

#### 7.435.2.6 Real gaussianPotentialUpside (Real *percentile*) const

gaussian-assumption Potential-Upside at a given percentile

**Precondition:**

percentile must be in range [90-100%)

#### 7.435.2.7 Real gaussianValueAtRisk (Real *percentile*) const

gaussian-assumption Value-At-Risk at a given percentile

**Precondition:**

percentile must be in range [90-100%)

**7.435.2.8 Real gaussianExpectedShortfall (Real *percentile*) const**

gaussian-assumption Expected Shortfall at a given percentile

Assuming a gaussian distribution it returns the expected loss in case that the loss exceeded a VaR threshold,

$$E[x \mid x < \text{VaR}(p)],$$

that is the average of observations below the given percentile  $p$ . Also know as conditional value-at-risk.

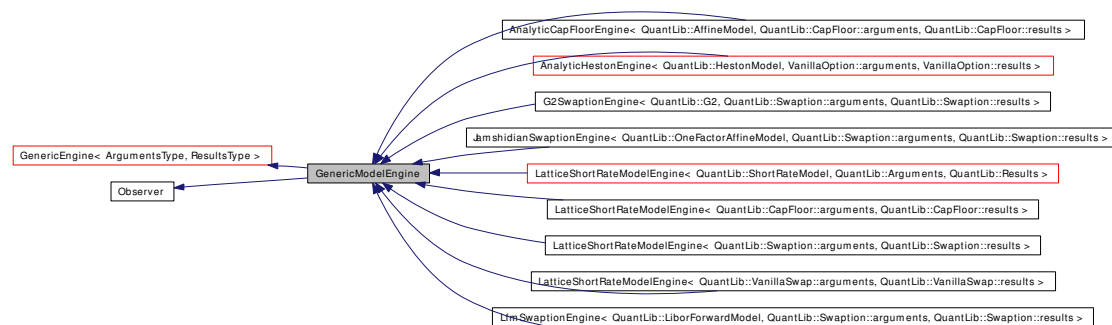
See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)



## 7.436 GenericModelEngine Class Template Reference

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Inheritance diagram for GenericModelEngine:



### 7.436.1 Detailed Description

```
template<class ModelType, class ArgumentsType, class ResultsType> class QuantLib::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 boost::shared\_ptr< ModelType > &model)
- void **setModel** (const boost::shared\_ptr< ModelType > &model)
- virtual void **update** ()

### Protected Attributes

- boost::shared\_ptr< ModelType > **model\_**

### 7.436.2 Member Function Documentation

#### 7.436.2.1 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](#).

Reimplemented in [LatticeShortRateModelEngine](#), [LatticeShortRateModelEngine< QuantLib::VanillaSwap::arguments, QuantLib::VanillaSwap::results >](#), [LatticeShortRateModelEngine< QuantLib::Swaption::arguments, QuantLib::Swaption::results >](#), and [LatticeShortRateModelEngine< QuantLib::CapFloor::arguments, QuantLib::CapFloor::results >](#).

## 7.437 GenericRiskStatistics Class Template Reference

```
#include <ql/Math/riskstatistics.hpp>
```

### 7.437.1 Detailed Description

```
template<class S> class QuantLib::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 statistic tool.

#### Todo

add historical annualized volatility

#### Examples:

[DiscreteHedging.cpp](#).

### Public Types

- typedef S::value\_type **value\_type**

### Public Member Functions

- Real [semiVariance](#) () const
- Real [semiDeviation](#) () const
- Real [downsideVariance](#) () const
- Real [downsideDeviation](#) () const
- Real [regret](#) (Real target) const
- Real [potentialUpside](#) (Real percentile) const  
*potential upside (the reciprocal of VAR) at a given percentile*
- Real [valueAtRisk](#) (Real percentile) const  
*value-at-risk at a given percentile*
- Real [expectedShortfall](#) (Real percentile) const  
*expected shortfall at a given percentile*
- Real [shortfall](#) (Real target) const
- Real [averageShortfall](#) (Real target) const

## 7.437.2 Member Function Documentation

### 7.437.2.1 Real semiVariance () const

returns the variance of observations below the mean,

$$\frac{N}{N-1} \mathbb{E} \left[ (x - \langle x \rangle)^2 \mid x < \langle x \rangle \right].$$

See Markowitz (1959).

### 7.437.2.2 Real semiDeviation () const

returns the semi deviation, defined as the square root of the semi variance.

### 7.437.2.3 Real downsideVariance () const

returns the variance of observations below 0.0,

$$\frac{N}{N-1} \mathbb{E} \left[ x^2 \mid x < 0 \right].$$

### 7.437.2.4 Real downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

### 7.437.2.5 Real regret (Real *target*) const

returns the variance of observations below target,

$$\frac{N}{N-1} \mathbb{E} \left[ (x - t)^2 \mid x < t \right].$$

See Dembo and Freeman, "The Rules Of Risk", Wiley (2001).

### 7.437.2.6 Real potentialUpside (Real *centile*) const

potential upside (the reciprocal of VAR) at a given percentile

**Precondition:**

percentile must be in range [90-100%)

### 7.437.2.7 Real valueAtRisk (Real *centile*) const

value-at-risk at a given percentile

**Precondition:**

percentile must be in range [90-100%)

**7.437.2.8 Real expectedShortfall (Real *percentile*) const**

expected shortfall at a given percentile

returns the expected loss in case that the loss exceeded a VaR threshold,

$$E[x \mid x < \text{VaR}(p)],$$

that is the average of observations below the given percentile  $p$ . Also known as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

**7.437.2.9 Real shortfall (Real *target*) const**

probability of missing the given target, defined as

$$E[\Theta \mid (-\infty, \infty)]$$

where

$$\Theta(x) = \begin{cases} 1 & x < t \\ 0 & x \geq t \end{cases}$$

**7.437.2.10 Real averageShortfall (Real *target*) const**

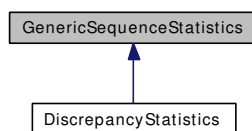
averaged shortfallness, defined as

$$E[t - x \mid x < t]$$

## 7.438 GenericSequenceStatistics Class Template Reference

```
#include <ql/Math/sequencestatistics.hpp>
```

Inheritance diagram for GenericSequenceStatistics:



### 7.438.1 Detailed Description

```
template<class StatisticsType = Statistics> class QuantLib::GenericSequenceStatistics<
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.

#### Tests

the correctness of the returned values is tested by checking them against numerical calculations.

### Public Types

- typedef StatisticsType **statistics\_type**
- typedef std::vector< typename StatisticsType::value\_type > **value\_type**

### Public Member Functions

- **GenericSequenceStatistics** (Size dimension)

#### inspectors

- Size **size** () const

#### covariance and correlation

- [Disposable](#)< [Matrix](#) > **covariance** () const  
*returns the covariance [Matrix](#)*
- [Disposable](#)< [Matrix](#) > **correlation** () const

returns the correlation [Matrix](#)

### 1-D inspectors lifted from underlying statistics class

- Size **samples** () const
- Real **weightSum** () const

### N-D inspectors lifted from underlying statistics class

- std::vector< Real > **mean** () const
- std::vector< Real > **variance** () const
- std::vector< Real > **standardDeviation** () const
- std::vector< Real > **downsideVariance** () const
- std::vector< Real > **downsideDeviation** () const
- std::vector< Real > **semiVariance** () const
- std::vector< Real > **semiDeviation** () const
- std::vector< Real > **errorEstimate** () const
- std::vector< Real > **skewness** () const
- std::vector< Real > **kurtosis** () const
- std::vector< Real > **min** () const
- std::vector< Real > **max** () const
- std::vector< Real > **gaussianPercentile** (Real y) const
- std::vector< Real > **percentile** (Real y) const
- std::vector< Real > **gaussianPotentialUpside** (Real percentile) const
- std::vector< Real > **potentialUpside** (Real percentile) const
- std::vector< Real > **gaussianValueAtRisk** (Real percentile) const
- std::vector< Real > **valueAtRisk** (Real percentile) const
- std::vector< Real > **gaussianExpectedShortfall** (Real percentile) const
- std::vector< Real > **expectedShortfall** (Real percentile) const
- std::vector< Real > **regret** (Real target) const
- std::vector< Real > **gaussianShortfall** (Real target) const
- std::vector< Real > **shortfall** (Real target) const
- std::vector< Real > **gaussianAverageShortfall** (Real target) const
- std::vector< Real > **averageShortfall** (Real target) const

### Modifiers

- void **reset** (Size dimension=0)
- template<class Sequence> void **add** (const Sequence &sample, Real weight=1.0)
- template<class Iterator> void **add** (Iterator begin, Iterator end, Real weight=1.0)

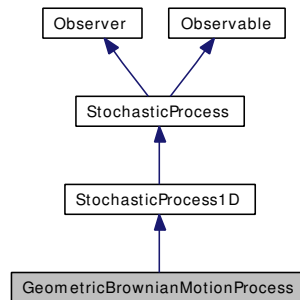
### Protected Attributes

- Size **dimension\_**
- std::vector< statistics\_type > **stats\_**
- std::vector< Real > **results\_**
- [Matrix](#) **quadraticSum\_**

## 7.439 GeometricBrownianMotionProcess Class Reference

```
#include <ql/Processes/geometricbrownianprocess.hpp>
```

Inheritance diagram for GeometricBrownianMotionProcess:



### 7.439.1 Detailed Description

Geometric brownian-motion process.

This class describes the stochastic process governed by

$$dS(t, S) = \mu S dt + \sigma S dW_t.$$

### Public Member Functions

- **GeometricBrownianMotionProcess** (double initialValue, double mue, double sigma)
- Real **x0** () const  
*returns the initial value of the state variable*
- Real **drift** (Time t, Real x) const  
*returns the drift part of the equation, i.e.  $\mu(t, x_t)$*
- Real **diffusion** (Time t, Real x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*

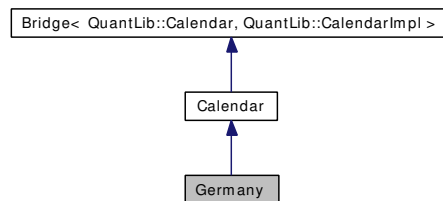
### Protected Attributes

- double **initialValue\_**
- double **mue\_**
- double **sigma\_**

## 7.440 Germany Class Reference

```
#include <ql/Calendars/germany.hpp>
```

Inheritance diagram for Germany:



### 7.440.1 Detailed Description

German calendars.

Public 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

Holidays for the Frankfurt [Stock](http://deutsche-boerse.com/) exchange (data from <http://deutsche-boerse.com/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday



- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

Holidays for the Xetra exchange (data from <http://deutsche-boerse.com/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

Holidays for the Eurex exchange (data from <http://www.eurexexchange.com/index.html>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

## Tests

the correctness of the returned results is tested against a list of known holidays.

## Public Types

- enum [Market](#) { [Settlement](#), [FrankfurtStockExchange](#), [Xetra](#), [Eurex](#) }  
*German calendars.*

## Public Member Functions

- [Germany](#) ([Market](#) market=FrankfurtStockExchange)

## 7.440.2 Member Enumeration Documentation

### 7.440.2.1 enum [Market](#)

German calendars.

#### Enumerator:

*Settlement* generic settlement calendar

*FrankfurtStockExchange* Frankfurt stock-exchange.

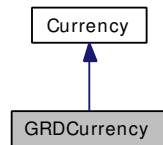
*Xetra* Xetra.

*Eurex* Eurex.

## 7.441 GRDCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for GRDCurrency:



### 7.441.1 Detailed Description

Greek drachma.

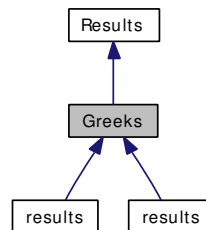
The ISO three-letter code was GRD; the numeric code was 300. It was divided in 100 lepta.

Obsoleted by the Euro since 2001.

## 7.442 Greeks Class Reference

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

Inheritance diagram for Greeks:



### 7.442.1 Detailed Description

additional option results

#### Public Member Functions

- void **reset** ()

#### Public Attributes

- Real **delta**
- Real **gamma**
- Real **theta**
- Real **vega**
- Real **rho**
- Real **dividendRho**

## 7.443 HaltonRsg Class Reference

```
#include <ql/RandomNumbers/haltonrsg.hpp>
```

### 7.443.1 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

#### Tests

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

### Public Types

- typedef [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

## 7.444 Handle Class Template Reference

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

### 7.444.1 Detailed Description

**template<class T> class QuantLib::Handle< T >**

Globally accessible relinkable pointer.

An instance of this class can be relinked to another shared pointer: such change will be propagated to all the copies of the instance.

#### Precondition:

Class T must inherit from [Observable](#)

#### Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), [FRA.cpp](#), [Replication.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

### Public Member Functions

- [Handle](#) (const boost::shared\_ptr< T > &h=boost::shared\_ptr< T >(), bool registerAsObserver=true)
- void [linkTo](#) (const boost::shared\_ptr< T > &, bool registerAsObserver=true)
- const boost::shared\_ptr< T > & [currentLink](#) () const  
*dereferencing*
- const boost::shared\_ptr< T > & **operator** → () const
- bool [empty](#) () const  
*checks if the contained shared pointer points to anything*
- [operator](#) boost::shared\_ptr () const  
*allows registration as observable*
- void [swap](#) ([Handle](#)< T > &other)  
*swaps two handles*
- template<class U> bool [operator==](#) (const [Handle](#)< U > &other)  
*equality test*
- template<class U> bool [operator!=](#) (const [Handle](#)< U > &other)  
*disequality test*
- template<class U> bool [operator<](#) (const [Handle](#)< U > &other)  
*strict weak ordering*

## Related Functions

(Note that these are not member functions.)

- void **swap** ([Handle](#)< T > &, [Handle](#)< T > &)

## 7.444.2 Constructor & Destructor Documentation

7.444.2.1 [Handle](#) (const boost::shared\_ptr< T > & *h* = boost::shared\_ptr< T >(), bool *registerAsObserver* = true) [explicit]

### Warning

see the documentation of the [Link](#) class for issues relatives to registerAsObserver.

## 7.444.3 Member Function Documentation

7.444.3.1 void **linkTo** (const boost::shared\_ptr< T > &, bool *registerAsObserver* = true)

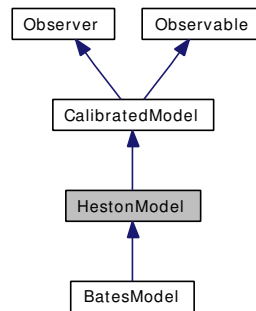
### Warning

see the documentation of the [Link](#) class for issues relatives to registerAsObserver.

## 7.445 HestonModel Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp>
```

Inheritance diagram for HestonModel:



### 7.445.1 Detailed Description

Heston model for the stochastic volatility of an asset.

References:

Heston, Steven L., 1993. A Closed-Form Solution for Options with Stochastic Volatility with Applications to [Bond](#) and [Currency](#) Options. The review of Financial Studies, Volume 6, Issue 2, 327-343.

#### Tests

calibration is tested against known good values.

### Public Member Functions

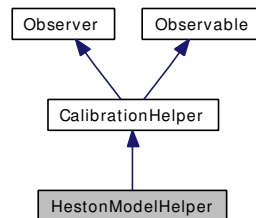
- **HestonModel** (const boost::shared\_ptr< [HestonProcess](#) > &process)
- [Real](#) **theta** () const
- [Real](#) **kappa** () const
- [Real](#) **sigma** () const
- [Real](#) **rho** () const
- [Real](#) **v0** () const
- boost::shared\_ptr< [NumericalMethod](#) > **tree** (const [TimeGrid](#) &) const



## 7.446 HestonModelHelper Class Reference

```
#include <ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp>
```

Inheritance diagram for HestonModelHelper:



### 7.446.1 Detailed Description

calibration helper for Heston model

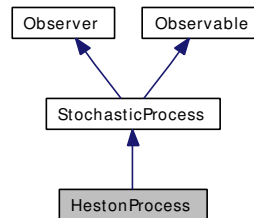
#### Public Member Functions

- **HestonModelHelper** (const [Period](#) &maturity, const [Calendar](#) &calendar, const Real s0, const Real strikePrice, const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [YieldTermStructure](#) > &dividendYield, bool calibrateVolatility=false)
- void **addTimesTo** (std::list< Time > &) const
- Real **modelValue** () const  
*returns the price of the instrument according to the model*
- Real **blackPrice** (Real volatility) const
- Time **maturity** () const

## 7.447 HestonProcess Class Reference

```
#include <ql/Processes/hestonprocess.hpp>
```

Inheritance diagram for HestonProcess:



### 7.447.1 Detailed Description

Square-root stochastic-volatility Heston process.

This class describes the square root stochastic volatility process governed by

$$\begin{aligned}
 dS(t, S) &= \mu S dt + \sqrt{v} S dW_1 \\
 dv(t, S) &= \kappa(\theta - v) dt + \sigma \sqrt{v} dW_2 \\
 dW_1 dW_2 &= \rho dt
 \end{aligned}$$

### Public Member Functions

- **HestonProcess** (const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [YieldTermStructure](#) > &dividendYield, const [Handle](#)< [Quote](#) > &s0, Real v0, Real kappa, Real theta, Real sigma, Real rho)
- [Size](#) [size](#) () const  
*returns the number of dimensions of the stochastic process*
- [Disposable](#)< [Array](#) > [initialValues](#) () const  
*returns the initial values of the state variables*
- [Disposable](#)< [Array](#) > [drift](#) (Time t, const [Array](#) &x) const  
*returns the drift part of the equation, i.e.,  $\mu(t, x_t)$*
- [Disposable](#)< [Matrix](#) > [diffusion](#) (Time t, const [Array](#) &x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*
- [Disposable](#)< [Array](#) > [apply](#) (const [Array](#) &x0, const [Array](#) &dx) const
- Real [s0](#) () const
- Real [v0](#) () const
- Real [rho](#) () const
- Real [kappa](#) () const
- Real [theta](#) () const
- Real [sigma](#) () const
- const boost::shared\_ptr< [YieldTermStructure](#) > & [dividendYield](#) () const
- const boost::shared\_ptr< [YieldTermStructure](#) > & [riskFreeRate](#) () const
- Time [time](#) (const [Date](#) &) const

## 7.447.2 Member Function Documentation

### 7.447.2.1 `Disposable<Array> apply (const Array & x0, const Array & dx) const` [virtual]

applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented from [StochasticProcess](#).

### 7.447.2.2 `Time time (const Date &) const` [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

**Note:**

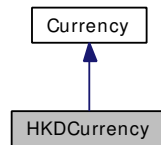
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

## 7.448 HKDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for HKDCurrency:



### 7.448.1 Detailed Description

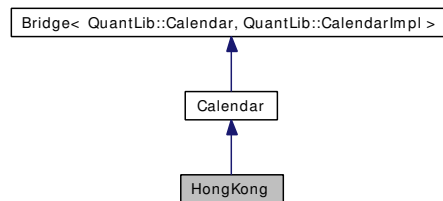
Honk Kong dollar.

The ISO three-letter code is HKD; the numeric code is 344. It is divided in 100 cents.

## 7.449 HongKong Class Reference

```
#include <ql/Calendars/hongkong.hpp>
```

Inheritance diagram for HongKong:



### 7.449.1 Detailed Description

Hong Kong calendars.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Ching Ming Festival, April 5th
- Good Friday
- Easter Monday
- Labor Day, May 1st
- SAR Establishment Day, July 1st
- National Day, October 1st (possibly moved to Monday)
- Christmas, December 25th
- Boxing Day, December 26th (possibly moved to Monday)

Other holidays for which no rule is given (data available for 2004-2006 only:)

- Lunar New Year
- Chinese New Year
- Buddha's birthday
- Tuen NG Festival
- Mid-autumn Festival
- Chung Yeung Festival

Data from <http://www.hkex.com.hk>

## Public Types

- enum [Market](#) { [HKEx](#) }

## Public Member Functions

- [HongKong](#) ([Market](#) m=[HKEx](#))

## 7.449.2 Member Enumeration Documentation

### 7.449.2.1 enum [Market](#)

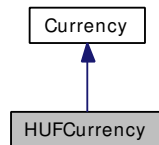
Enumerator:

*HKEx* Hong Kong stock exchange.

## 7.450 HUFCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for HUFCurrency:



### 7.450.1 Detailed Description

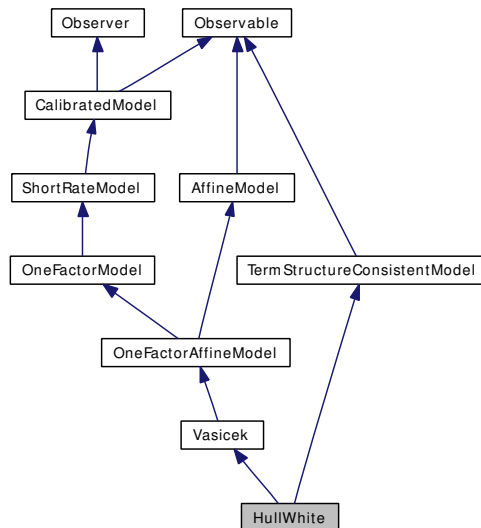
Hungarian forint.

The ISO three-letter code is HUF; the numeric code is 348. It has no subdivisions.

## 7.451 HullWhite Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/hullwhite.hpp>
```

Inheritance diagram for HullWhite:



### 7.451.1 Detailed Description

Single-factor Hull-White (extended Vasicek) 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  $\alpha$  and  $\sigma$  are constants.

#### Tests

calibration results are tested against cached values

#### Bug

When the term structure is relinked, the `r0` parameter of the underlying [Vasicek](#) model is not updated.

#### Examples:

[BermudanSwaption.cpp](#).

### Public Member Functions

- **HullWhite** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, Real a=0.1, Real sigma=0.01)
- `boost::shared_ptr< NumericalMethod > tree` (const [TimeGrid](#) &grid) const  
*Return by default a trinomial recombining tree.*



- `boost::shared_ptr< ShortRateDynamics > dynamics () const`  
*returns the short-rate dynamics*
- Real `discountBondOption` (Option::Type type, Real strike, Time maturity, Time bond-Maturity) const

## Static Public Member Functions

- static `Rate convexityBias` (Real futurePrice, Time t, Time T, Real sigma, Real a)

## Protected Member Functions

- void `generateArguments` ()
- Real A (Time t, Time T) const

## Classes

- class `Dynamics`  
*Short-rate dynamics in the Hull-White model.*
- class `FittingParameter`  
*Analytical term-structure fitting parameter  $\varphi(t)$ .*

## 7.451.2 Member Function Documentation

### 7.451.2.1 static `Rate convexityBias` (Real *futurePrice*, Time *t*, Time *T*, Real *sigma*, Real *a*) [static]

Futures convexity bias (i.e., the difference between futures implied rate and forward rate) calculated as in G. Kirikos, D. Novak, "Convexity Conundrums", Risk Magazine, March 1997.

#### Note:

t and T should be expressed in yearfraction using deposit day counter, F\_quoted is futures' market price.

## 7.452 HullWhite::Dynamics Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/hullwhite.hpp>
```

### 7.452.1 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.

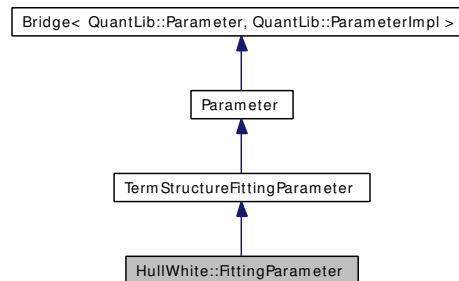
### Public Member Functions

- **Dynamics** (const [Parameter](#) &fitting, [Real](#) a, [Real](#) sigma)
- **Real variable** ([Time](#) t, [Rate](#) r) const
- **Real shortRate** ([Time](#) t, [Real](#) x) const

## 7.453 HullWhite::FittingParameter Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/hullwhite.hpp>
```

Inheritance diagram for HullWhite::FittingParameter:



### 7.453.1 Detailed Description

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

$\varphi(t)$  is analytically defined by

$$\varphi(t) = f(t) + \frac{1}{2} \left[ \frac{\sigma(1 - e^{-at})}{a} \right]^2,$$

where  $f(t)$  is the instantaneous forward rate at  $t$ .

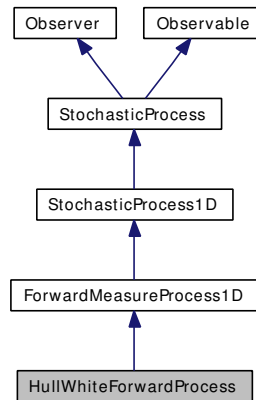
### Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a, [Real](#) sigma)

## 7.454 HullWhiteForwardProcess Class Reference

```
#include <ql/Processes/hullwhiteprocess.hpp>
```

Inheritance diagram for HullWhiteForwardProcess:



### 7.454.1 Detailed Description

forward Hull-White stochastic process

### Public Member Functions

- **HullWhiteForwardProcess** (const [Handle](#)< [YieldTermStructure](#) > &h, Real a, Real sigma)

#### StochasticProcess1D interface

- Real [x0](#) () const  
*returns the initial value of the state variable*
- Real [drift](#) (Time t, Real x) const  
*returns the drift part of the equation, i.e.  $\mu(t, x_t)$*
- Real [diffusion](#) (Time t, Real x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*
- Real [expectation](#) (Time t0, Real x0, Time dt) const
- Real [stdDeviation](#) (Time t0, Real x0, Time dt) const
- Real [variance](#) (Time t0, Real x0, Time dt) const

### Protected Member Functions

- Real [alpha](#) (Time t) const
- Real [M\\_T](#) (Real s, Real t, Real T) const
- Real [B](#) (Time t, Time T) const

## Protected Attributes

- `boost::shared_ptr< QuantLib::OrnsteinUhlenbeckProcess > process_`
- `Handle< YieldTermStructure > h_`
- Real `a_`
- Real `sigma_`

## 7.454.2 Member Function Documentation

### 7.454.2.1 Real expectation (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

returns the expectation  $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

### 7.454.2.2 Real stdDeviation (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

returns the standard deviation  $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

### 7.454.2.3 Real variance (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

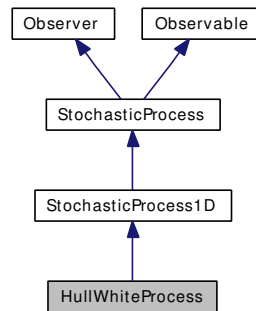
returns the variance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

## 7.455 HullWhiteProcess Class Reference

```
#include <ql/Processes/hullwhiteprocess.hpp>
```

Inheritance diagram for HullWhiteProcess:



### 7.455.1 Detailed Description

Hull-White stochastic process.

#### Public Member Functions

- **HullWhiteProcess** (const [Handle](#)< [YieldTermStructure](#) > &h, Real a, Real sigma)

#### StochasticProcess1D interface

- Real [x0](#) () const  
*returns the initial value of the state variable*
- Real [drift](#) (Time t, Real x) const  
*returns the drift part of the equation, i.e.  $\mu(t, x_i)$*
- Real [diffusion](#) (Time t, Real x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_i)$*
- Real [expectation](#) (Time t0, Real x0, Time dt) const
- Real [stdDeviation](#) (Time t0, Real x0, Time dt) const
- Real [variance](#) (Time t0, Real x0, Time dt) const

#### Protected Member Functions

- Real [alpha](#) (Time t) const

#### Protected Attributes

- boost::shared\_ptr< [QuantLib::OrnsteinUhlenbeckProcess](#) > **process\_**
- [Handle](#)< [YieldTermStructure](#) > **h\_**
- Real **a\_**
- Real **sigma\_**

## 7.455.2 Member Function Documentation

### 7.455.2.1 Real expectation (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

returns the expectation  $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

### 7.455.2.2 Real stdDeviation (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

returns the standard deviation  $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

### 7.455.2.3 Real variance (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

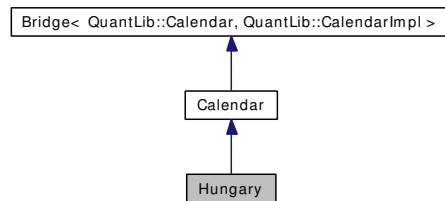
returns the variance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

## 7.456 Hungary Class Reference

```
#include <ql/Calendars/hungary.hpp>
```

Inheritance diagram for Hungary:



### 7.456.1 Detailed Description

Hungarian calendar.

Holidays:

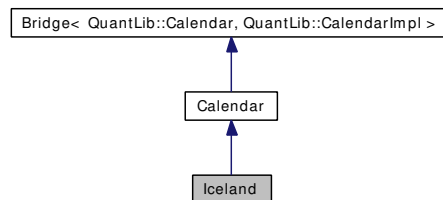
- Saturdays
- Sundays
- Easter Monday
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- National Day, March 15th
- Labour Day, May 1st
- Constitution Day, August 20th
- Republic Day, October 23rd
- All Saints Day, November 1st
- Christmas, December 25th
- 2nd Day of Christmas, December 26th



## 7.457 Iceland Class Reference

```
#include <ql/Calendars/iceland.hpp>
```

Inheritance diagram for Iceland:



### 7.457.1 Detailed Description

Icelandic calendars.

Holidays for the [Iceland](http://www.icex.is/is/calendar?language-ID=1) stock exchange (data from [<http://www.icex.is/is/calendar?language-ID=1>](http://www.icex.is/is/calendar?language-ID=1)):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Holy Thursday
- Good Friday
- Easter Monday
- First day of Summer (third or fourth Thursday in April)
- Labour Day, May 1st
- Ascension Thursday
- Pentecost Monday
- Independence Day, June 17th
- Commerce Day, first Monday in August
- Christmas, December 25th
- Boxing Day, December 26th

### Public Types

- enum [Market](#) { [ICEX](#) }

### Public Member Functions

- [Iceland](#) ([Market](#) m=[ICEX](#))

## 7.457.2 Member Enumeration Documentation

### 7.457.2.1 enum [Market](#)

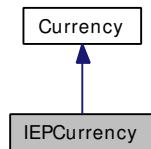
Enumerator:

*ICEX* [Iceland](#) stock exchange.

## 7.458 IEPCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for IEPCurrency:



### 7.458.1 Detailed Description

Irish punt.

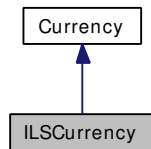
The ISO three-letter code was IEP; the numeric code was 372. It was divided in 100 pence.

Obsoleted by the Euro since 1999.

## 7.459 ILSCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for ILSCurrency:



### 7.459.1 Detailed Description

Israeli shekel.

The ISO three-letter code is ILS; the numeric code is 376. It is divided in 100 agorot.

## 7.460 IMM Struct Reference

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

### 7.460.1 Detailed Description

Main cycle of the International [Money](#) Market (a.k.a. [IMM](#)) Months.

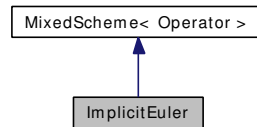
#### Public Types

- enum **Month** {  
    **F** = 1, **G** = 2, **H** = 3, **J** = 4,  
    **K** = 5, **M** = 6, **N** = 7, **Q** = 8,  
    **U** = 9, **V** = 10, **X** = 11, **Z** = 12 }

## 7.461 ImplicitEuler Class Template Reference

```
#include <ql/FiniteDifferences/impliciteuler.hpp>
```

Inheritance diagram for ImplicitEuler:



### 7.461.1 Detailed Description

```
template<class Operator> class QuantLib::ImplicitEuler< Operator >
```

Backward Euler scheme for finite difference methods.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```

typedef ... array_type;

// 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
array_type solveFor(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);

```

### Public Types

- typedef OperatorTraits< Operator > **traits**
- typedef traits::operator\_type **operator\_type**
- typedef traits::array\_type **array\_type**
- typedef traits::bc\_set **bc\_set**
- typedef [traits::condition\\_type](#) **condition\_type**

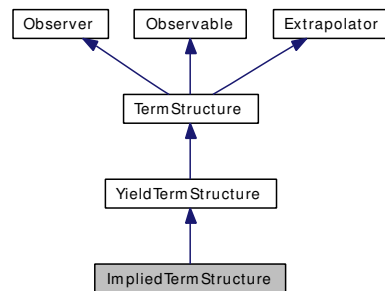
### Public Member Functions

- **ImplicitEuler** (const operator\_type &L, const bc\_set &bcs)

## 7.462 ImpliedTermStructure Class Reference

```
#include <ql/TermStructures/impliedtermstructure.hpp>
```

Inheritance diagram for ImpliedTermStructure:



### 7.462.1 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.

#### Tests

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure is checked.

### Public Member Functions

- **ImpliedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Date](#) &reference-Date)

#### YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Calendar](#) [calendar](#) () const  
*the calendar used for reference date calculation*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return values*

## Protected Member Functions

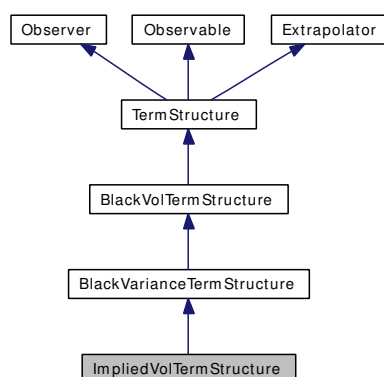
- [DiscountFactor](#) `discountImpl` (Time) const  
*returns the discount factor as seen from the evaluation date*



## 7.463 ImpliedVolTermStructure Class Reference

```
#include <ql/Volatilities/impliedvoltermstructure.hpp>
```

Inheritance diagram for ImpliedVolTermStructure:



### 7.463.1 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 [Handle](#)< [BlackVolTermStructure](#) > &originalTS, const [Date](#) &referenceDate)

#### BlackVolTermStructure interface

- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return values*
- Real [minStrike](#) () const  
*the minimum strike for which the term structure can return vols*
- Real [maxStrike](#) () const

*the maximum strike for which the term structure can return vols*

### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

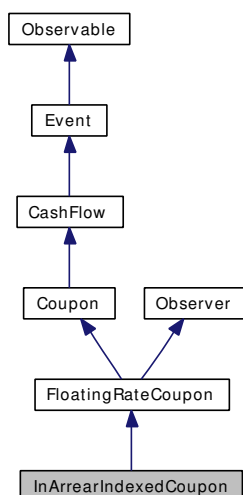
### Protected Member Functions

- virtual Real [blackVarianceImpl](#) (Time t, Real strike) const  
*Black variance calculation.*

## 7.464 InArrearIndexedCoupon Class Reference

#include <ql/CashFlows/inarrearindexedcoupon.hpp>

Inheritance diagram for InArrearIndexedCoupon:



### 7.464.1 Detailed Description

In-arrear floating-rate 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.

#### Tests

The class is tested by comparing the value of an in-arrear swap against a known good value.

### Public Member Functions

- **InArrearIndexedCoupon** (const [Date](#) &paymentDate, const [Real](#) nominal, const [Date](#) &startDate, const [Date](#) &endDate, const [Integer](#) fixingDays, const boost::shared\_ptr< [Xibor](#) > &index, const [Real](#) gearing=1.0, const [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())

#### FloatingRateCoupon interface

- [Date](#) **fixingDate** () const  
*fixing date*

#### Modifiers

- void **setCapletVolatility** (const [Handle](#)< [CapletVolatilityStructure](#) > &)

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

#### Protected Member Functions

- Rate [convexityAdjustmentImpl](#) (Rate fixing) const  
*convexity adjustment for the given index fixing*

#### Protected Attributes

- [Handle](#)< [CapletVolatilityStructure](#) > **capletVolatility\_**

## 7.465 IncrementalStatistics Class Reference

```
#include <ql/Math/incrementalstatistics.hpp>
```

### 7.465.1 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 Types

- typedef Real **value\_type**

### Public Member Functions

#### Inspectors

- Size **samples** () const  
*number of samples collected*
- Real **weightSum** () const  
*sum of data weights*
- Real **mean** () const
- Real **variance** () const
- Real **standardDeviation** () const
- Real **errorEstimate** () const
- Real **skewness** () const
- Real **kurtosis** () const
- Real **min** () const
- Real **max** () const
- Real **downsideVariance** () const
- Real **downsideDeviation** () const

#### Modifiers

- void **add** (Real value, Real 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_`
- Real `sampleWeight_`
- Real `downsideSampleWeight_`
- Real `sum_`
- Real `quadraticSum_`
- Real `downsideQuadraticSum_`
- Real `cubicSum_`
- Real `fourthPowerSum_`
- Real `min_`
- Real `max_`

## 7.465.2 Member Function Documentation

### 7.465.2.1 Real `mean () const`

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

### 7.465.2.2 Real `variance () const`

returns the variance, defined as

$$\frac{N}{N-1} \langle (x - \langle x \rangle)^2 \rangle.$$

### 7.465.2.3 Real `standardDeviation () const`

returns the standard deviation  $\sigma$ , defined as the square root of the variance.

### 7.465.2.4 Real `errorEstimate () const`

returns the error estimate  $\epsilon$ , defined as the square root of the ratio of the variance to the number of samples.

### 7.465.2.5 Real `skewness () const`

returns the skewness, defined as

$$\frac{N^2}{(N-1)(N-2)} \frac{\langle (x - \langle x \rangle)^3 \rangle}{\sigma^3}.$$

The above evaluates to 0 for a Gaussian distribution.

**7.465.2.6 Real kurtosis () const**

returns the excess kurtosis, defined as

$$\frac{N^2(N+1)}{(N-1)(N-2)(N-3)} \frac{\langle (x - \langle x \rangle)^4 \rangle}{\sigma^4} - \frac{3(N-1)^2}{(N-2)(N-3)}.$$

The above evaluates to 0 for a Gaussian distribution.

**7.465.2.7 Real min () const**

returns the minimum sample value

**7.465.2.8 Real max () const**

returns the maximum sample value

**7.465.2.9 Real 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  $\theta = 0$  if  $x > 0$  and  $\theta = 1$  if  $x < 0$

**7.465.2.10 Real downsideDeviation () const**

returns the downside deviation, defined as the square root of the downside variance.

**7.465.2.11 void add (Real value, Real weight = 1.0)**

adds a datum to the set, possibly with a weight

**Precondition:**

weight must be positive or null

**7.465.2.12 void addSequence (DataIterator begin, DataIterator end, WeightIterator wbegin)**

adds a sequence of data to the set, each with its weight

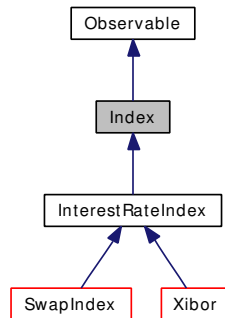
**Precondition:**

weights must be positive or null

## 7.466 Index Class Reference

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

Inheritance diagram for Index:



### 7.466.1 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, bool forecastTodaysFixing=false) const=0  
*returns the fixing at the given date*
- void [addFixing](#) (const [Date](#) &fixingDate, [Rate](#) fixing)  
*stores the historical fixing at the given date*
- template<class DateIterator, class ValueIterator> void [addFixings](#) (DateIterator dBegin, DateIterator dEnd, ValueIterator vBegin)  
*stores historical fixings at the given dates*
- void [clearFixings](#) ()  
*clears all stored historical fixings*

### 7.466.2 Member Function Documentation

#### 7.466.2.1 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.



**Todo**

add methods returning [InterestRate](#)

Implemented in [InterestRateIndex](#).

**7.466.2.2** `virtual Rate fixing (const Date & fixingDate, bool forecastTodaysFixing = false)  
const [pure virtual]`

returns the fixing at the given date

the date passed as arguments must be the actual calendar date of the fixing; no settlement days must be used.

Implemented in [InterestRateIndex](#).

**7.466.2.3** `void addFixing (const Date & fixingDate, Rate fixing)`

stores the historical fixing at the given date

the date passed as arguments must be the actual calendar date of the fixing; no settlement days must be used.

**7.466.2.4** `void addFixings (DateIterator dBegin, DateIterator dEnd, ValueIterator vBegin)`

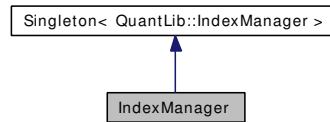
stores historical fixings at the given dates

the dates passed as arguments must be the actual calendar dates of the fixings; no settlement days must be used.

## 7.467 IndexManager Class Reference

```
#include <ql/Indexes/indexmanager.hpp>
```

Inheritance diagram for IndexManager:



### 7.467.1 Detailed Description

global repository for past index fixings

**Note:**

index names are case insensitive

### Public Member Functions

- bool **hasHistory** (const std::string &name) const  
*returns whether historical fixings were stored for the index*
- const **TimeSeries**< **Real** > & **getHistory** (const std::string &name) const  
*returns the (possibly empty) history of the index fixings*
- void **setHistory** (const std::string &name, const **TimeSeries**< **Real** > &)  
*stores the historical fixings of the index*
- boost::shared\_ptr< **Observable** > **notifier** (const std::string &name) const  
*observer notifying of changes in the index fixings*
- std::vector< std::string > **histories** () const  
*returns all names of the indexes for which fixings were stored*
- void **clearHistory** (const std::string &name)  
*clears the historical fixings of the index*
- void **clearHistories** ()  
*clears all stored fixings*

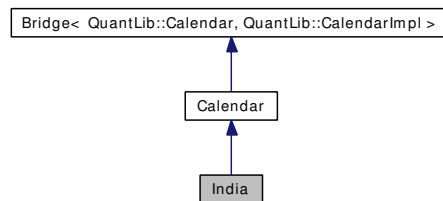
### Friends

- class **Singleton**< **IndexManager** >

## 7.468 India Class Reference

```
#include <ql/Calendars/india.hpp>
```

Inheritance diagram for India:



### 7.468.1 Detailed Description

Indian calendars.

Holidays for the National [Stock Exchange](http://www.nse-india.com/) (data from <http://www.nse-india.com/>):

- Saturdays
- Sundays
- Republic Day, January 26th
- Good Friday
- Ambedkar Jayanti, April 14th
- Independence Day, August 15th
- Gandhi Jayanti, October 2nd
- Christmas, December 25th

Other holidays for which no rule is given (data available for 2005 only:)

- Bakri Id
- Moharram
- Holi
- Maharashtra Day
- Ganesh Chaturthi
- Dasara
- Laxmi Puja
- Bhaubeej
- Ramzan Id
- Guru Nanak Jayanti

## Public Types

- enum [Market](#) { [NSE](#) }

## Public Member Functions

- [India](#) ([Market](#) m=NSE)

## 7.468.2 Member Enumeration Documentation

### 7.468.2.1 enum [Market](#)

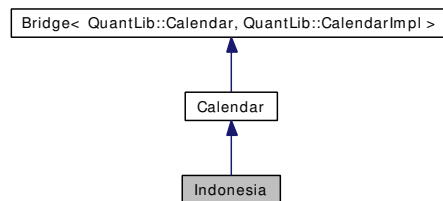
#### Enumerator:

[NSE](#) National [Stock](#) Exchange.

## 7.469 Indonesia Class Reference

```
#include <ql/Calendars/indonesia.hpp>
```

Inheritance diagram for Indonesia:



### 7.469.1 Detailed Description

Indonesian calendars

Holidays for the Jakarta stock exchange (data from <http://www.jsx.co.id/trading.asp?cmd=menu3>):

- Saturdays
- Sundays
- Good Friday
- Ascension of Jesus Christ
- Independence Day, August 17th
- Christmas, December 25th

Other holidays for which no rule is given (data available for 2005-2006 only:)

- Idul Adha
- Imlek
- Moslem's New Year Day
- Nyepi (Saka's New Year)
- Birthday of Prophet Muhammad SAW
- Waisak
- Ascension of Prophet Muhammad SAW
- Idul Fitri
- Other national leaves

### Public Types

- enum [Market](#) { [BEJ](#) }

## Public Member Functions

- Indonesia ([Market](#) m=BEJ)

## 7.469.2 Member Enumeration Documentation

### 7.469.2.1 enum [Market](#)

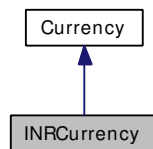
Enumerator:

*BEJ* Jakarta stock exchange.

## 7.470 INRCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for INRCurrency:



### 7.470.1 Detailed Description

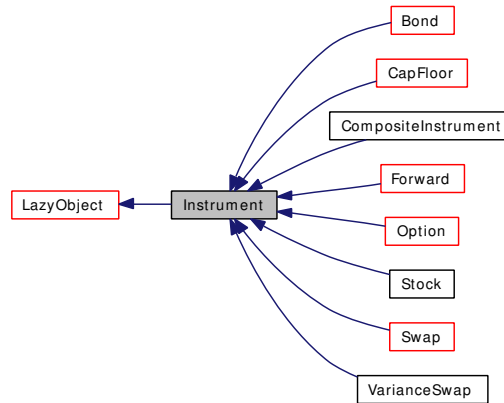
Indian rupee.

The ISO three-letter code is INR; the numeric code is 356. It is divided in 100 paise.

## 7.471 Instrument Class Reference

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

Inheritance diagram for Instrument:



### 7.471.1 Detailed Description

Abstract instrument class.

This class is purely abstract and defines the interface of concrete instruments which will be derived from this one.

#### Tests

observability of class instances is checked.

### Public Member Functions

- virtual void [setupArguments](#) ([Arguments](#) \*) const
- virtual void [fetchResults](#) (const [Results](#) \*) const

#### Inspectors

- Real [NPV](#) () const  
*returns the net present value of the instrument.*
- Real [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 boost::shared\_ptr< [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

- boost::shared\_ptr< [PricingEngine](#) > [engine\\_](#)

### Results

*The value of this attribute and any other that derived classes might declare must be set during calculation.*

- Real [NPV\\_](#)
- Real [errorEstimate\\_](#)

## 7.471.2 Member Function Documentation

### 7.471.2.1 void setPricingEngine (const boost::shared\_ptr< [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.

### 7.471.2.2 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 [ContinuousAveragingAsianOption](#), [DiscreteAveragingAsianOption](#), [AssetSwap](#), [BarrierOption](#), [BasketOption](#), [CapFloor](#), [CliquetOption](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [ContinuousFloatingLookbackOption](#), [ContinuousFixedLookbackOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoForwardVanillaOption](#), [QuantoVanillaOption](#), [Swaption](#), [VanillaSwap](#), and [VarianceSwap](#).

### 7.471.2.3 void fetchResults (const [Results](#) \*) const [virtual]

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

Reimplemented in [AssetSwap](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), [VanillaSwap](#), and [VarianceSwap](#).

#### 7.471.2.4 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.

##### Warning

Should this method be redefined in derived classes, [LazyObject::calculate\(\)](#) should be called in the overriding method.

Reimplemented from [LazyObject](#).

#### 7.471.2.5 void setupExpired () const [protected, virtual]

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

Reimplemented in [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), [Swap](#), and [VarianceSwap](#).

#### 7.471.2.6 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](#).

Reimplemented in [Bond](#), [CompositeInstrument](#), [FixedCouponBondForward](#), [Forward](#), [Forward-RateAgreement](#), [Stock](#), [Swap](#), and [VarianceSwap](#).

## 7.472 IntegralEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/integralengine.hpp>
```

### 7.472.1 Detailed Description

Pricing engine for European vanilla options using integral approach

#### Todo

define tolerance for calculate()

#### Examples:

[EquityOption.cpp](#).

### Public Member Functions

- void **calculate** () const

## 7.473 InterestRate Class Reference

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

### 7.473.1 Detailed Description

Concrete interest rate class.

This class encapsulate the interest rate compounding algebra. It manages day-counting conventions, compounding conventions, conversion between different conventions, discount/compound factor calculations, and implied/equivalent rate calculations.

#### Tests

Converted rates are checked against known good results

### Public Member Functions

#### conversions

- **operator Rate ()** const

#### inspectors

- Rate **rate ()** const
- const [DayCounter](#) & **dayCounter ()** const
- Compounding **compounding ()** const
- [Frequency](#) **frequency ()** const

#### discount/compound factor calculations

- [DiscountFactor](#) **discountFactor (Time t)** const  
*discount factor implied by the rate compounded at time t.*
- [DiscountFactor](#) **discountFactor (const [Date](#) &d1, const [Date](#) &d2, const [Date](#) &refStart=[Date](#)(), const [Date](#) &refEnd=[Date](#)())** const  
*discount factor implied by the rate compounded between two dates*
- Real [compoundFactor](#) (**Time t**) const  
*compound factor implied by the rate compounded at time t.*
- Real [compoundFactor](#) (const [Date](#) &d1, const [Date](#) &d2, const [Date](#) &refStart=[Date](#)(), const [Date](#) &refEnd=[Date](#)()) const  
*compound factor implied by the rate compounded between two dates*

#### equivalent rate calculations

- [InterestRate](#) **equivalentRate (Time t, Compounding comp, [Frequency](#) freq=Annual)** const  
*equivalent interest rate for a compounding period t.*
- [InterestRate](#) **equivalentRate (Date d1, Date d2, const [DayCounter](#) &resultDC, Compounding comp, [Frequency](#) freq=Annual)** const  
*equivalent rate for a compounding period between two dates*

## Static Public Member Functions

### implied rate calculations

- static [InterestRate](#) [impliedRate](#) (Real compound, [Time](#) t, const [DayCounter](#) &resultDC, Compounding comp, [Frequency](#) freq=Annual)  
*implied interest rate for a given compound factor at a given time.*
- static [InterestRate](#) [impliedRate](#) (Real compound, const [Date](#) &d1, const [Date](#) &d2, const [DayCounter](#) &resultDC, Compounding comp, [Frequency](#) freq=Annual)  
*implied rate for a given compound factor between two dates.*

## Related Functions

(Note that these are not member functions.)

- `std::ostream & operator<< (std::ostream &, const InterestRate &)`

## 7.473.2 Member Function Documentation

### 7.473.2.1 [DiscountFactor](#) [discountFactor](#) ([Time](#) t) const

discount factor implied by the rate compounded at time t.

#### Warning

Time must be measured using [InterestRate](#)'s own day counter.

### 7.473.2.2 Real compoundFactor ([Time](#) t) const

compound factor implied by the rate compounded at time t.

returns the compound (a.k.a capitalization) factor implied by the rate compounded at time t.

#### Warning

Time must be measured using [InterestRate](#)'s own day counter.

### 7.473.2.3 Real compoundFactor (const [Date](#) & d1, const [Date](#) & d2, const [Date](#) & refStart = [Date](#)(), const [Date](#) & refEnd = [Date](#)()) const

compound factor implied by the rate compounded between two dates

returns the compound (a.k.a capitalization) factor implied by the rate compounded between two dates.

7.473.2.4 **static [InterestRate](#) impliedRate (Real *compound*, [Time](#) *t*, const [DayCounter](#) & *resultDC*, [Compounding](#) *comp*, [Frequency](#) *freq* = Annual) [static]**

implied interest rate for a given compound factor at a given time.

The resulting [InterestRate](#) has the day-counter provided as input.

#### Warning

Time must be measured using the day-counter provided as input.

7.473.2.5 **static [InterestRate](#) impliedRate (Real *compound*, const [Date](#) & *d1*, const [Date](#) & *d2*, const [DayCounter](#) & *resultDC*, [Compounding](#) *comp*, [Frequency](#) *freq* = Annual) [static]**

implied rate for a given compound factor between two dates.

The resulting rate is calculated taking the required day-counting rule into account.

7.473.2.6 **[InterestRate](#) equivalentRate ([Time](#) *t*, [Compounding](#) *comp*, [Frequency](#) *freq* = Annual) const**

equivalent interest rate for a compounding period *t*.

The resulting [InterestRate](#) shares the same implicit day-counting rule of the original [InterestRate](#) instance.

#### Warning

Time must be measured using the [InterestRate](#)'s own day counter.

7.473.2.7 **[InterestRate](#) equivalentRate ([Date](#) *d1*, [Date](#) *d2*, const [DayCounter](#) & *resultDC*, [Compounding](#) *comp*, [Frequency](#) *freq* = Annual) const**

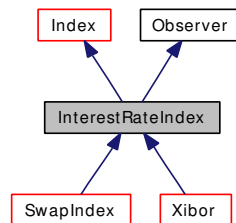
equivalent rate for a compounding period between two dates

The resulting rate is calculated taking the required day-counting rule into account.

## 7.474 InterestRateIndex Class Reference

```
#include <ql/Indexes/interestrateindex.hpp>
```

Inheritance diagram for InterestRateIndex:



### 7.474.1 Detailed Description

base class for interest rate indexes

#### Todo

add methods returning [InterestRate](#)

### Public Member Functions

- **InterestRateIndex** (const std::string &familyName, const [Period](#) &tenor, Integer settlement-Days, const [Currency](#) &currency, const [Calendar](#) &calendar, const [DayCounter](#) &day-Counter)

#### Index interface

- std::string [name](#) () const  
*Returns the name of the index.*
- [Rate fixing](#) (const [Date](#) &fixingDate, bool forecastTodaysFixing=false) const  
*returns the fixing at the given date*

#### Observer interface

- void [update](#) ()

#### Inspectors

- std::string [familyName](#) () const
- [Period](#) [tenor](#) () const
- Integer [settlementDays](#) () const
- const [Currency](#) & [currency](#) () const
- [Calendar](#) [calendar](#) () const
- const [DayCounter](#) & [dayCounter](#) () const
- virtual [Rate](#) [forecastFixing](#) (const [Date](#) &fixingDate) const=0
- virtual boost::shared\_ptr< [YieldTermStructure](#) > [termStructure](#) () const=0

### Date calculations

*These methods can be overridden to implement particular conventions*

- virtual [Date](#) **valueDate** (const [Date](#) &fixingDate) const
- virtual [Date](#) **maturityDate** (const [Date](#) &valueDate) const

### Protected Attributes

- std::string **familyName\_**
- [Period](#) **tenor\_**
- Integer **settlementDays\_**
- [Currency](#) **currency\_**
- [Calendar](#) **calendar\_**
- [DayCounter](#) **dayCounter\_**

## 7.474.2 Member Function Documentation

### 7.474.2.1 `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.

#### Todo

add methods returning [InterestRate](#)

Implements [Index](#).

### 7.474.2.2 `Rate fixing (const Date &fixingDate, bool forecastTodaysFixing = false) const` [virtual]

returns the fixing at the given date

the date passed as arguments must be the actual calendar date of the fixing; no settlement days must be used.

Implements [Index](#).

### 7.474.2.3 `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.

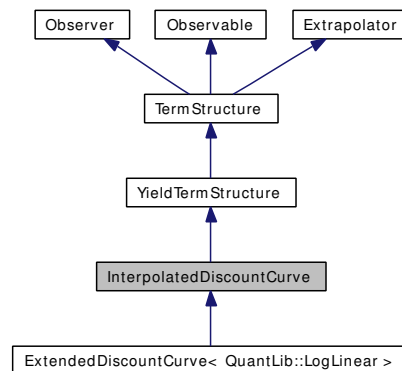
Implements [Observer](#).



## 7.475 InterpolatedDiscountCurve Class Template Reference

```
#include <ql/TermStructures/discountcurve.hpp>
```

Inheritance diagram for InterpolatedDiscountCurve:



### 7.475.1 Detailed Description

```
template<class Interpolator> class QuantLib::InterpolatedDiscountCurve< Interpolator >
```

Term structure based on interpolation of discount factors.

### Public Member Functions

- **InterpolatedDiscountCurve** (const std::vector< [Date](#) > &dates, const std::vector< DiscountFactor > &dfs, const [DayCounter](#) &dayCounter, const Interpolator &interpolator=Interpolator())

### Inspectors

- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- Time **maxTime** () const  
*the latest time for which the curve can return values*
- const std::vector< Time > & **times** () const
- const std::vector< [Date](#) > & **dates** () const
- const std::vector< DiscountFactor > & **discounts** () const
- std::vector< std::pair< [Date](#), DiscountFactor > > **nodes** () const

### Protected Member Functions

- **InterpolatedDiscountCurve** (const [DayCounter](#) &, const Interpolator &interpolator=Interpolator())

- **InterpolatedDiscountCurve** (const [Date](#) &referenceDate, const [DayCounter](#) &, const Interpolator &interpolator=Interpolator())
- **InterpolatedDiscountCurve** (Integer settlementDays, const [Calendar](#) &, const [DayCounter](#) &, const Interpolator &interpolator=Interpolator())
- DiscountFactor [discountImpl](#) (Time) const  
*discount calculation*

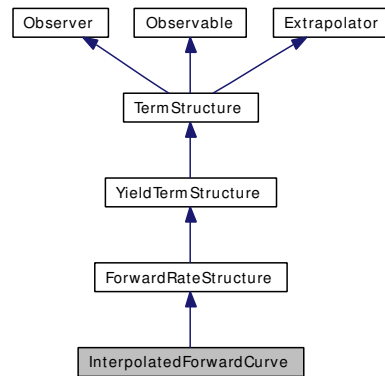
## Protected Attributes

- [DayCounter](#) **dayCounter\_**
- std::vector< [Date](#) > **dates\_**
- std::vector< Time > **times\_**
- std::vector< DiscountFactor > **data\_**
- [Interpolation](#) **interpolation\_**
- Interpolator **interpolator\_**

## 7.476 InterpolatedForwardCurve Class Template Reference

```
#include <ql/TermStructures/forwardcurve.hpp>
```

Inheritance diagram for InterpolatedForwardCurve:



### 7.476.1 Detailed Description

```
template<class Interpolator> class QuantLib::InterpolatedForwardCurve< Interpolator >
```

Term structure based on interpolation of forward rates.

#### Inspectors

- [DayCounter](#) **dayCounter\_**
- `std::vector< Date >` **dates\_**
- `std::vector< Time >` **times\_**
- `std::vector< Rate >` **data\_**
- [Interpolation](#) **interpolation\_**
- `Interpolator` **interpolator\_**
- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- `Time` **maxTime** () const  
*the latest time for which the curve can return values*
- `const std::vector< Time > &` **times** () const
- `const std::vector< Date > &` **dates** () const
- `const std::vector< Rate > &` **forwards** () const
- `std::vector< std::pair< Date, Rate > >` **nodes** () const
- **InterpolatedForwardCurve** (const [DayCounter](#) &, const `Interpolator` & **interpolator**=`Interpolator`())
- **InterpolatedForwardCurve** (const [Date](#) & **referenceDate**, const [DayCounter](#) &, const `Interpolator` & **interpolator**=`Interpolator`())

- **InterpolatedForwardCurve** (Integer settlementDays, const [Calendar](#) &, const [DayCounter](#) &, const Interpolator &interpolator=Interpolator())
- Rate [forwardImpl](#) (Time t) const  
*instantaneous forward-rate calculation*
- Rate [zeroYieldImpl](#) (Time t) const

## Public Member Functions

- **InterpolatedForwardCurve** (const std::vector< [Date](#) > &dates, const std::vector< Rate > &forwards, const [DayCounter](#) &dayCounter, const Interpolator &interpolator=Interpolator())

## 7.476.2 Member Function Documentation

### 7.476.2.1 Rate [zeroYieldImpl](#) (Time t) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

#### [Warning](#)

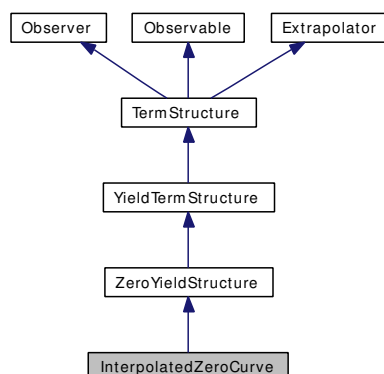
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own `zeroYield` method.

Reimplemented from [ForwardRateStructure](#).

## 7.477 InterpolatedZeroCurve Class Template Reference

```
#include <ql/TermStructures/zerocurve.hpp>
```

Inheritance diagram for InterpolatedZeroCurve:



### 7.477.1 Detailed Description

```
template<class Interpolator> class QuantLib::InterpolatedZeroCurve< Interpolator >
```

Term structure based on interpolation of zero yields.

#### Inspectors

- [DayCounter](#) **dayCounter\_**
- `std::vector< Date >` **dates\_**
- `std::vector< Time >` **times\_**
- `std::vector< Rate >` **data\_**
- [Interpolation](#) **interpolation\_**
- `Interpolator` **interpolator\_**
- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- `Time` **maxTime** () const  
*the latest time for which the curve can return values*
- `const std::vector< Time > &` **times** () const
- `const std::vector< Date > &` **dates** () const
- `const std::vector< Rate > &` **zeroRates** () const
- `std::vector< std::pair< Date, Rate > >` **nodes** () const
- **InterpolatedZeroCurve** (const [DayCounter](#) &, const `Interpolator` & **interpolator**=`Interpolator`())
- **InterpolatedZeroCurve** (const [Date](#) & **referenceDate**, const [DayCounter](#) &, const `Interpolator` & **interpolator**=`Interpolator`())

- **InterpolatedZeroCurve** (Integer settlementDays, const [Calendar](#) &, const [DayCounter](#) &, const Interpolator &interpolator=Interpolator())
- Rate [zeroYieldImpl](#) (Time t) const  
*zero-yield calculation*

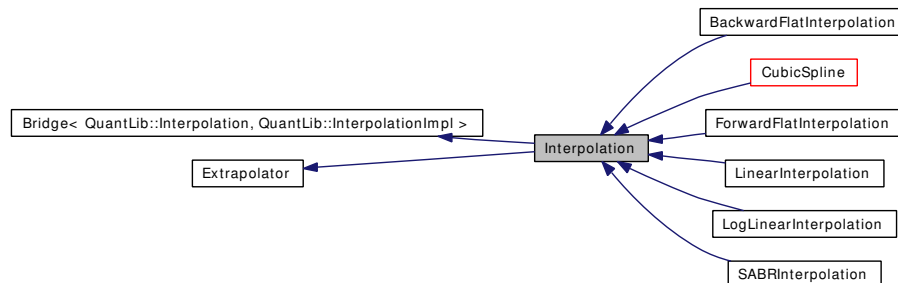
## Public Member Functions

- **InterpolatedZeroCurve** (const std::vector< [Date](#) > &dates, const std::vector< Rate > &yields, const [DayCounter](#) &dayCounter, const Interpolator &interpolator=Interpolator())

## 7.478 Interpolation Class Reference

```
#include <ql/Math/interpolation.hpp>
```

Inheritance diagram for Interpolation:



### 7.478.1 Detailed Description

base class for 1-D interpolations.

Classes derived from this class will provide interpolated values from two sequences of equal length, representing discretized values of a variable and a function of the former, respectively.

### Public Types

- typedef [Real](#) `argument_type`
- typedef [Real](#) `result_type`

### Public Member Functions

- [Real](#) `operator() (Real x, bool allowExtrapolation=false) const`
- [Real](#) `primitive (Real x, bool allowExtrapolation=false) const`
- [Real](#) `derivative (Real x, bool allowExtrapolation=false) const`
- [Real](#) `secondDerivative (Real x, bool allowExtrapolation=false) const`
- [Real](#) `xMin () const`
- [Real](#) `xMax () const`
- [bool](#) `isInRange (Real x) const`
- `void` `update ()`

### Protected Member Functions

- `void` `checkRange (Real x, bool extrapolate) const`

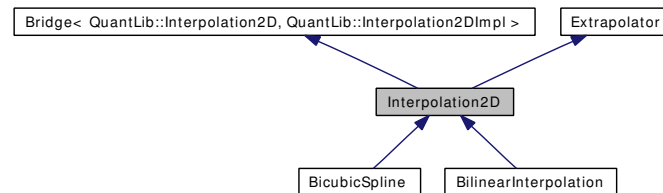
### Classes

- class [templateImpl](#)  
*basic template implementation*

## 7.479 Interpolation2D Class Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2D:



### 7.479.1 Detailed Description

base class for 2-D interpolations.

Classes derived from this class will provide interpolated values from two sequences of length  $N$  and  $M$ , representing the discretized values of the  $x$  and  $y$  variables, and a  $N \times M$  matrix representing the tabulated function values.

### Public Types

- typedef [Real](#) **first\_argument\_type**
- typedef [Real](#) **second\_argument\_type**
- typedef [Real](#) **result\_type**

### Public Member Functions

- [Real](#) **operator()** ([Real](#) x, [Real](#) y, bool allowExtrapolation=false) const
- [Real](#) **xMin** () const
- [Real](#) **xMax** () const
- std::vector< [Real](#) > **xValues** () const
- Size **locateX** ([Real](#) x) const
- [Real](#) **yMin** () const
- [Real](#) **yMax** () const
- std::vector< [Real](#) > **yValues** () const
- Size **locateY** ([Real](#) y) const
- const [Matrix](#) & **zData** () const
- bool **isInRange** ([Real](#) x, [Real](#) y) const
- void **update** ()

### Protected Member Functions

- void **checkRange** ([Real](#) x, [Real](#) y, bool extrapolate) const



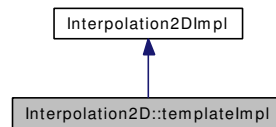
## Classes

- class [templateImpl](#)  
*basic template implementation*

## 7.480 Interpolation2D::templateImpl Class Template Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2D::templateImpl:



### 7.480.1 Detailed Description

```
template<class I1, class I2, class M> class QuantLib::Interpolation2D::templateImpl< I1, I2, M
>
```

basic template implementation

### Public Member Functions

- **templateImpl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)
- Real **xMin** () const
- Real **xMax** () const
- std::vector< Real > **xValues** () const
- Real **yMin** () const
- Real **yMax** () const
- std::vector< Real > **yValues** () const
- const [Matrix](#) & **zData** () const
- bool **isInRange** (Real x, Real y) const

### Protected Member Functions

- Size **locateX** (Real x) const
- Size **locateY** (Real y) const

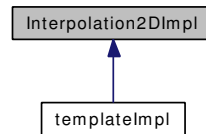
### Protected Attributes

- I1 **xBegin\_**
- I1 **xEnd\_**
- I2 **yBegin\_**
- I2 **yEnd\_**
- const M & **zData\_**

## 7.481 Interpolation2DImpl Class Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2DImpl:



### 7.481.1 Detailed Description

abstract base class for 2-D interpolation implementations

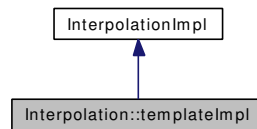
#### Public Member Functions

- virtual void **calculate** ()=0
- virtual Real **xMin** () const=0
- virtual Real **xMax** () const=0
- virtual std::vector< Real > **xValues** () const=0
- virtual Size **locateX** (Real x) const=0
- virtual Real **yMin** () const=0
- virtual Real **yMax** () const=0
- virtual std::vector< Real > **yValues** () const=0
- virtual Size **locateY** (Real y) const=0
- virtual const **Matrix** & **zData** () const=0
- virtual bool **isInRange** (Real x, Real y) const=0
- virtual Real **value** (Real x, Real y) const=0

## 7.482 Interpolation::templateImpl Class Template Reference

```
#include <ql/Math/interpolation.hpp>
```

Inheritance diagram for Interpolation::templateImpl:



### 7.482.1 Detailed Description

```
template<class I1, class I2> class QuantLib::Interpolation::templateImpl< I1, I2 >
```

basic template implementation

#### Public Member Functions

- **templateImpl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)
- Real **xMin** () const
- Real **xMax** () const
- bool **isInRange** (Real x) const

#### Protected Member Functions

- Size **locate** (Real x) const

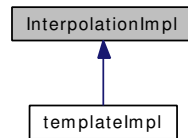
#### Protected Attributes

- I1 **xBegin\_**
- I1 **xEnd\_**
- I2 **yBegin\_**

## 7.483 InterpolationImpl Class Reference

```
#include <ql/Math/interpolation.hpp>
```

Inheritance diagram for InterpolationImpl:



### 7.483.1 Detailed Description

abstract base class for interpolation implementations

#### Public Member Functions

- virtual void **calculate** ()=0
- virtual Real **xMin** () const=0
- virtual Real **xMax** () const=0
- virtual bool **isInRange** (Real) const=0
- virtual Real **value** (Real) const=0
- virtual Real **primitive** (Real) const=0
- virtual Real **derivative** (Real) const=0
- virtual Real **secondDerivative** (Real) const=0

## 7.484 IntervalPrice Class Reference

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

### 7.484.1 Detailed Description

interval price

#### Public Types

- enum **Type** { **Open**, **Close**, **High**, **Low** }

#### Public Member Functions

- **IntervalPrice** (Real open, Real close, Real high, Real low)

##### Inspectors

- Real **open** () const
- Real **close** () const
- Real **high** () const
- Real **low** () const
- Real **value** (IntervalPrice::Type) const

##### Modifiers

- void **setValue** (Real value, IntervalPrice::Type)
- void **setValues** (Real open, Real close, Real high, Real low)

#### Static Public Member Functions

##### Helper functions

- static [TimeSeries](#)< [IntervalPrice](#) > **makeSeries** (const std::vector< [Date](#) > &d, const std::vector< Real > &open, const std::vector< Real > &close, const std::vector< Real > &high, const std::vector< Real > &low)
- static std::vector< Real > **extractValues** (const [TimeSeries](#)< [IntervalPrice](#) > &, IntervalPrice::Type)
- static [TimeSeries](#)< Real > **extractComponent** (const [TimeSeries](#)< [IntervalPrice](#) > &, enum IntervalPrice::Type)

## 7.485 InverseCumulativeNormal Class Reference

```
#include <ql/Math/normaldistribution.hpp>
```

### 7.485.1 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** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real  $x$ ) const

## 7.486 InverseCumulativePoisson Class Reference

```
#include <ql/Math/poissondistribution.hpp>
```

### 7.486.1 Detailed Description

Inverse cumulative Poisson distribution function.

#### Tests

the correctness of the returned value is tested by checking it against known good results.

### Public Member Functions

- **InverseCumulativePoisson** (Real lambda=1.0)
- Real **operator()** (Real x) const



## 7.487 InverseCumulativeRng Class Template Reference

```
#include <ql/RandomNumbers/inversecumulativrng.hpp>
```

### 7.487.1 Detailed Description

```
template<class RNG, class IC> class QuantLib::InverseCumulativeRng< RNG, IC >
```

Inverse cumulative random number generator.

It uses a uniform deviate in (0, 1) as the source of cumulative distribution values. Then an inverse cumulative distribution is used to calculate the distribution deviate.

The uniform deviate is supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

The inverse cumulative distribution is supplied by IC.

Class IC must implement the following interface:

```
IC::IC();  
Real IC::operator() const;
```

### Public Types

- typedef [Sample](#)< [Real](#) > **sample\_type**
- typedef RNG **urng\_type**

### Public Member Functions

- **InverseCumulativeRng** (const RNG &uniformGenerator)
- [sample\\_type](#) **next** () const  
*returns a sample from a Gaussian distribution*

## 7.488 InverseCumulativeRsg Class Template Reference

```
#include <ql/RandomNumbers/inversecumulativersg.hpp>
```

### 7.488.1 Detailed Description

```
template<class USG, class IC> class QuantLib::InverseCumulativeRsg< USG, IC >
```

Inverse cumulative random sequence generator.

It uses a sequence of uniform deviate in (0, 1) as the source of cumulative distribution values. Then an inverse cumulative distribution is used to calculate the distribution deviate.

The uniform deviate sequence is supplied by USG.

Class USG must implement the following interface:

```
USG::sample_type USG::nextSequence() const;
Size USG::dimension() const;
```

The inverse cumulative distribution is supplied by IC.

Class IC must implement the following interface:

```
IC::IC();
Real IC::operator() const;
```

Examples:

[DiscreteHedging.cpp](#).

### Public Types

- typedef [Sample< Array >](#) **sample\_type**

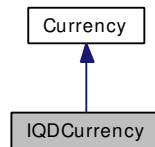
### Public Member Functions

- **InverseCumulativeRsg** (const USG &uniformSequenceGenerator)
- **InverseCumulativeRsg** (const USG &uniformSequenceGenerator, const IC &inverseCumulative)
- const [sample\\_type](#) & [nextSequence](#) () const  
*returns next sample from the Gaussian distribution*
- const [sample\\_type](#) & [lastSequence](#) () const
- Size [dimension](#) () const

## 7.489 IQDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for IQDCurrency:



### 7.489.1 Detailed Description

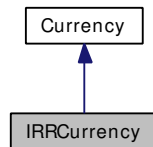
Iraqi dinar.

The ISO three-letter code is IQD; the numeric code is 368. It is divided in 1000 fils.

## 7.490 IRRCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for IRRCurrency:



### 7.490.1 Detailed Description

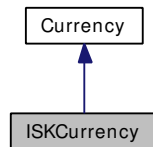
Iranian rial.

The ISO three-letter code is IRR; the numeric code is 364. It has no subdivisions.

## 7.491 ISKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ISKCurrency:



### 7.491.1 Detailed Description

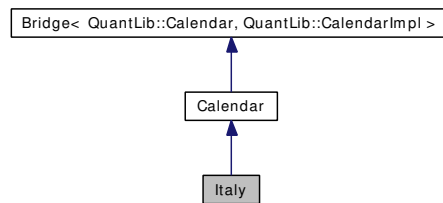
[Iceland](#) krona.

The ISO three-letter code is ISK; the numeric code is 352. It is divided in 100 aurar.

## 7.492 Italy Class Reference

```
#include <ql/Calendars/italy.hpp>
```

Inheritance diagram for Italy:



### 7.492.1 Detailed Description

Italian calendars.

Public 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 Day, December 8th
- Christmas Day, December 25th
- St. Stephen's Day, December 26th

Holidays for the stock exchange (data from <http://www.borsaitalia.it>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday

- Labour Day, May 1st
- Assumption, August 15th
- Christmas' Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th
- New Year's Eve, December 31st

### Tests

the correctness of the returned results is tested against a list of known holidays.

### Public Types

- enum [Market](#) { [Settlement](#), [Exchange](#) }
- Italian calendars.*

### Public Member Functions

- Italy ([Market](#) market=[Settlement](#))

## 7.492.2 Member Enumeration Documentation

### 7.492.2.1 enum [Market](#)

Italian calendars.

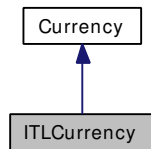
#### Enumerator:

*Settlement* generic settlement calendar  
*Exchange* Milan stock-exchange calendar.

## 7.493 ITLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ITLCurrency:



### 7.493.1 Detailed Description

Italian lira.

The ISO three-letter code was ITL; the numeric code was 380. It had no subdivisions.

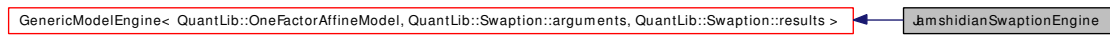
Obsoleted by the Euro since 1999.



## 7.494 JamshidianSwaptionEngine Class Reference

```
#include <ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp>
```

Inheritance diagram for JamshidianSwaptionEngine:



### 7.494.1 Detailed Description

Jamshidian swaption engine.

#### Warning

The engine assumes that the exercise date equals the start date of the passed swap.

#### Examples:

[BermudanSwaption.cpp](#).

### Public Member Functions

- **JamshidianSwaptionEngine** (const boost::shared\_ptr< [OneFactorAffineModel](#) > &model)
- void **calculate** () const

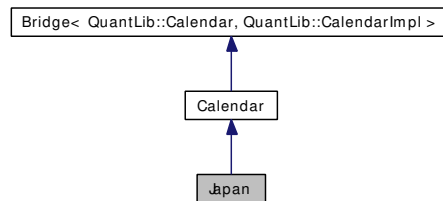
### Friends

- class **rStarFinder**

## 7.495 Japan Class Reference

```
#include <ql/Calendars/japan.hpp>
```

Inheritance diagram for Japan:



### 7.495.1 Detailed Description

Japanese 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

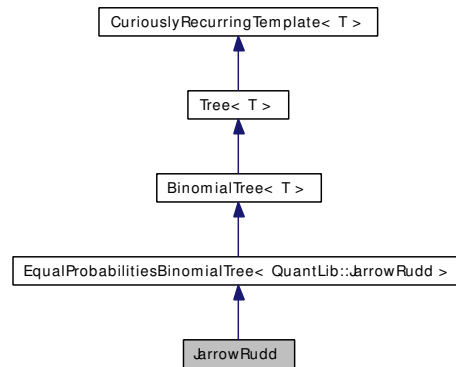
- a few one-shot holidays

Holidays falling on a Sunday are observed on the Monday following except for the bank holidays associated with the new year.

## 7.496 JarrowRudd Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for JarrowRudd:



### 7.496.1 Detailed Description

Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

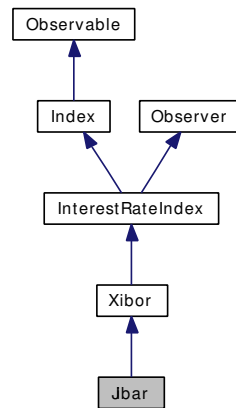
#### Public Member Functions

- **JarrowRudd** (const boost::shared\_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)

## 7.497 Jibar Class Reference

```
#include <ql/Indexes/jibar.hpp>
```

Inheritance diagram for Jibar:



### 7.497.1 Detailed Description

JIBAR rate

Johannesburg Interbank Agreed Rate

#### Todo

check settlement days and day-count convention.

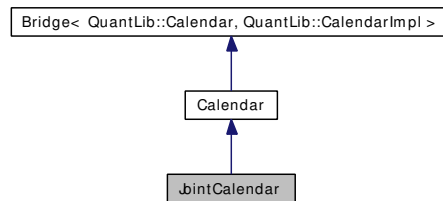
### Public Member Functions

- **Jibar** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.498 JointCalendar Class Reference

```
#include <ql/Calendars/jointcalendar.hpp>
```

Inheritance diagram for JointCalendar:



### 7.498.1 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.

#### Tests

the correctness of the returned results is tested by reproducing the calculations.

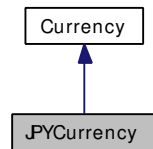
### Public Member Functions

- **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)

## 7.499 JPYCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for JPYCurrency:



### 7.499.1 Detailed Description

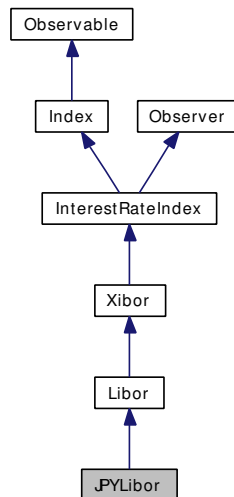
Japanese yen.

The ISO three-letter code is JPY; the numeric code is 392. It is divided into 100 sen.

## 7.500 JPYLibor Class Reference

```
#include <ql/Indexes/jpylibor.hpp>
```

Inheritance diagram for JPYLibor:



### 7.500.1 Detailed Description

JPY LIBOR rate

Japanese Yen LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

#### Warning

This is the rate fixed in London by BBA. Use TIBOR if you're interested in the Tokio fixing.

### Public Member Functions

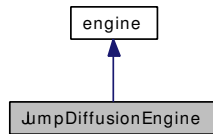
- **JPYLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference, [Integer](#) settlementDays=2)



## 7.501 JumpDiffusionEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp>
```

Inheritance diagram for JumpDiffusionEngine:



### 7.501.1 Detailed Description

Jump-diffusion engine for vanilla options.

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

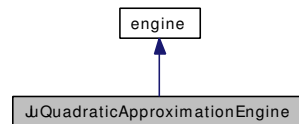
### Public Member Functions

- **JumpDiffusionEngine** (const boost::shared\_ptr< [VanillaOption::engine](#) > &, Real relative-Accuracy\_=1e-4, Size maxIterations=100)
- void **calculate** () const

## 7.502 JuQuadraticApproximationEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/juquadraticengine.hpp>
```

Inheritance diagram for JuQuadraticApproximationEngine:



### 7.502.1 Detailed Description

Pricing engine for American options with Ju quadratic approximation

An Approximate Formula for Pricing American Options Journal of Derivatives Winter 1999 Ju, N.

#### Warning

Barone-Adesi-Whaley critical commodity price calculation is used, it has not been modified to see whether the method of Ju is faster. Ju does not say how he solves the equation for the critical stock price, e.g. [Newton](#) method. He just gives the solution. The method of BAW gives answers to the same accuracy as in Ju (1999).

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

#### Bug

test fails for Borland compiler

### Public Member Functions

- void **calculate** () const

## 7.503 KnuthUniformRng Class Reference

```
#include <ql/RandomNumbers/knuthuniformrng.hpp>
```

### 7.503.1 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 [Sample< Real >](#) `sample_type`

### Public Member Functions

- [KnuthUniformRng](#) (long seed=0)
- [sample\\_type next](#) () const

### 7.503.2 Constructor & Destructor Documentation

#### 7.503.2.1 [KnuthUniformRng](#) (long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

### 7.503.3 Member Function Documentation

#### 7.503.3.1 [KnuthUniformRng::sample\\_type next](#) () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

## 7.504 KronrodIntegral Class Reference

```
#include <ql/Math/kronrodintegral.hpp>
```

### 7.504.1 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](http://www.math.iastate.edu/burkardt/f_src/nms/nms.html)>

#### Tests

the correctness of the result is tested by checking it against known good values.

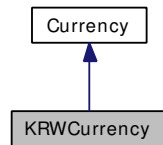
### Public Member Functions

- **KronrodIntegral** (Real tolerance, Size maxFunctionEvaluations=[Null](#)< Size >())
- template<class F> Real **operator()** (const F &f, Real a, Real b) const
- Size **functionEvaluations** ()

## 7.505 KRWCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for KRWCurrency:



### 7.505.1 Detailed Description

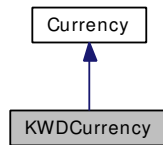
South-Korean won.

The ISO three-letter code is KRW; the numeric code is 410. It is divided in 100 chon.

## 7.506 KWDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for KWDCurrency:



### 7.506.1 Detailed Description

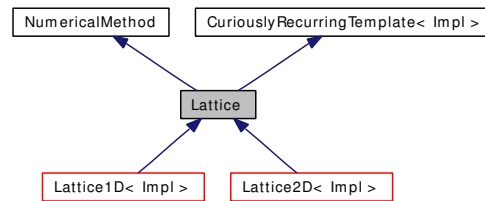
Kuwaiti dinar.

The ISO three-letter code is KWD; the numeric code is 414. It is divided in 1000 fils.

## 7.507 Lattice Class Template Reference

```
#include <ql/Lattices/lattice.hpp>
```

Inheritance diagram for Lattice:



### 7.507.1 Detailed Description

```
template<class Impl> class QuantLib::Lattice< Impl >
```

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.

Derived classes must implement the following interface:

```
public:
    DiscountFactor discount(Size i, Size index) const;
    Size descendant(Size i, Size index, Size branch) const;
    Real probability(Size i, Size index, Size branch) const;
```

and may implement the following:

```
public:
    void stepback(Size i,
                  const Array& values,
                  Array& newValues) const;
```

### Public Member Functions

- **Lattice** (const [TimeGrid](#) &timeGrid, Size n)
- const [Array](#) & **statePrices** (Size i) const
- void **stepback** (Size i, const [Array](#) &values, [Array](#) &newValues) const

#### NumericalMethod interface

- void **initialize** ([DiscretizedAsset](#) &, Time t) const  
*initialize an asset at the given time.*
- void **rollback** ([DiscretizedAsset](#) &, Time to) const
- void **partialRollback** ([DiscretizedAsset](#) &, Time to) const
- [Real](#) **presentValue** ([DiscretizedAsset](#) &) const  
*Computes the present value of an asset using Arrow-Debreu prices.*

## Protected Member Functions

- void `computeStatePrices` (Size until) const

## Protected Attributes

- std::vector< [Array](#) > `statePrices_`

## 7.507.2 Member Function Documentation

### 7.507.2.1 void `rollback` ([DiscretizedAsset](#) &, Time *to*) const [virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implements [NumericalMethod](#).

Reimplemented in [TsiveriotisFernandesLattice](#).

### 7.507.2.2 void `partialRollback` ([DiscretizedAsset](#) &, Time *to*) const [virtual]

Roll back an asset until the given time, but do not perform the final adjustment.

## Warning

In version 0.3.7 and earlier, this method was called `rollAlmostBack` method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Implements [NumericalMethod](#).

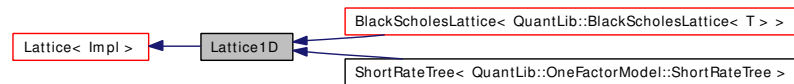
Reimplemented in [TsiveriotisFernandesLattice](#).



## 7.508 Lattice1D Class Template Reference

```
#include <ql/Lattices/lattice1d.hpp>
```

Inheritance diagram for Lattice1D:



### 7.508.1 Detailed Description

```
template<class Impl> class QuantLib::Lattice1D< Impl >
```

One-dimensional lattice.

Derived classes must implement the following interface:

```
Real underlying(Size i, Size index) const;
```

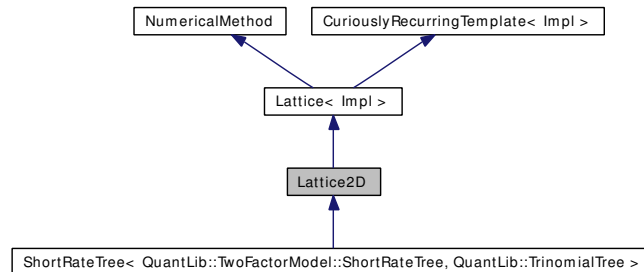
### Public Member Functions

- **Lattice1D** (const [TimeGrid](#) &timeGrid, Size n)
- **Disposable**< [Array](#) > **grid** (Time t) const
- **Real** **underlying** (Size i, Size index) const

## 7.509 Lattice2D Class Template Reference

```
#include <ql/Lattices/lattice2d.hpp>
```

Inheritance diagram for Lattice2D:



### 7.509.1 Detailed Description

```
template<class Impl, class T = TrinomialTree> class QuantLib::Lattice2D< Impl, T >
```

Two-dimensional lattice.

This lattice is based on two trinomial trees and primarily used for the [G2](#) short-rate model.

#### Public Member Functions

- **Lattice2D** (const boost::shared\_ptr< T > &tree1, const boost::shared\_ptr< T > &tree2, Real correlation)
- Size **size** (Size i) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

#### Protected Member Functions

- [Disposable](#)< [Array](#) > **grid** (Time) const

#### Protected Attributes

- boost::shared\_ptr< T > **tree1\_**
- boost::shared\_ptr< T > **tree2\_**

## 7.510 LatticeShortRateModelEngine Class Template Reference

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

Inheritance diagram for LatticeShortRateModelEngine:



### 7.510.1 Detailed Description

```
template<class Arguments, class Results> class QuantLib::LatticeShortRateModelEngine< Ar-
guments, 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 boost::shared\_ptr< [ShortRateModel](#) > &model, Size timeSteps)
- **LatticeShortRateModelEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &model, const [TimeGrid](#) &timeGrid)
- void [update](#) ()

### Protected Attributes

- [TimeGrid](#) `timeGrid_`
- Size `timeSteps_`
- boost::shared\_ptr< [NumericalMethod](#) > `lattice_`

### 7.510.2 Member Function Documentation

#### 7.510.2.1 void update () [virtual]

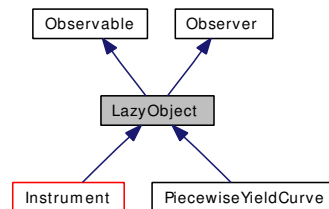
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [GenericModelEngine](#).

## 7.511 LazyObject Class Reference

```
#include <ql/Patterns/lazyobject.hpp>
```

Inheritance diagram for LazyObject:



### 7.511.1 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_`

### 7.511.2 Member Function Documentation

#### 7.511.2.1 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](#).

Reimplemented in [PiecewiseYieldCurve](#).

#### 7.511.2.2 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.

#### 7.511.2.3 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.

#### 7.511.2.4 void unfreeze ()

This method reverts the effect of the `freeze` method, thus re-enabling recalculations.

#### 7.511.2.5 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.

##### Warning

Should this method be redefined in derived classes, `LazyObject::calculate()` should be called in the overriding method.

Reimplemented in [Instrument](#).

#### 7.511.2.6 virtual void performCalculations () const [protected, pure virtual]

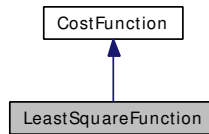
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implemented in [Instrument](#), [Bond](#), [CompositeInstrument](#), [FixedCouponBondForward](#), [Forward](#), [ForwardRateAgreement](#), [Stock](#), [Swap](#), and [VarianceSwap](#).

## 7.512 LeastSquareFunction Class Reference

```
#include <ql/Optimization/leastsquare.hpp>
```

Inheritance diagram for LeastSquareFunction:



### 7.512.1 Detailed Description

Cost function for least-square problems.

Implements a cost function using the interface provided by the [LeastSquareProblem](#) class.

#### Public Member Functions

- [LeastSquareFunction](#) ([LeastSquareProblem](#) &lsp)  
*Default constructor.*
- virtual [~LeastSquareFunction](#) ()  
*Destructor.*
- virtual [Real 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 [Real valueAndGradient](#) ([Array](#) &grad\_f, const [Array](#) &x) const  
*compute value and gradient of the least square function*

#### Protected Attributes

- [LeastSquareProblem](#) & lsp\_  
*least square problem*

## 7.513 LeastSquareProblem Class Reference

```
#include <ql/Optimization/leastsquare.hpp>
```

### 7.513.1 Detailed Description

Base class for least square problem.

#### Public Member Functions

- virtual `Size` `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 function to fit*
- virtual void `targetValueAndGradient` (const `Array` &x, `Matrix` &grad\_fct2fit, `Array` &target, `Array` &fct2fit)=0

### 7.513.2 Member Function Documentation

**7.513.2.1** virtual void `targetValueAndGradient` (const `Array` & x, `Matrix` & grad\_fct2fit, `Array` & target, `Array` & fct2fit) [pure virtual]

compute the target vector, the values of the function to fit and the matrix of derivatives

## 7.514 LecuyerUniformRng Class Reference

```
#include <ql/RandomNumbers/lecuyeruniformrng.hpp>
```

### 7.514.1 Detailed Description

Uniform random number generator.

Random number generator of L'Ecuyer with added Bays-Durham shuffle (known 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 [Sample< Real >](#) `sample_type`

### Public Member Functions

- [LecuyerUniformRng](#) (long seed=0)
- [sample\\_type next](#) () const

### 7.514.2 Constructor & Destructor Documentation

#### 7.514.2.1 [LecuyerUniformRng](#) (long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

### 7.514.3 Member Function Documentation

#### 7.514.3.1 [sample\\_type next](#) () const

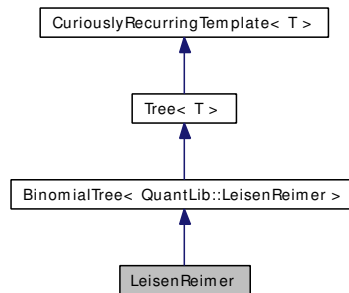
returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)



## 7.515 LeisenReimer Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for LeisenReimer:



### 7.515.1 Detailed Description

Leisen & Reimer tree: multiplicative approach.

#### Public Member Functions

- **LeisenReimer** (const boost::shared\_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

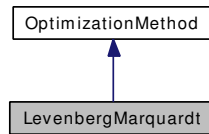
#### Protected Attributes

- Real **up\_**
- Real **down\_**
- Real **pu\_**
- Real **pd\_**

## 7.516 LevenbergMarquardt Class Reference

```
#include <ql/Optimization/levenbergmarquardt.hpp>
```

Inheritance diagram for LevenbergMarquardt:



### 7.516.1 Detailed Description

Levenberg-Marquardt optimization method.

This implementation is based on MINPACK (<http://www.netlib.org/minpack>), <http://www.netlib.org/cephes/linalg.tgz>)

Examples:

[BermudanSwaption.cpp](#).

### Public Member Functions

- [LevenbergMarquardt](#) (Real *epsfcn*=1e-8, Real *ftol*=1e-8, Real *xtol*=1e-8, Real *gtol*=1e-8, Size *maxfev*=400)
- void [minimize](#) (const [Problem](#) &P) const  
*minimize the optimization problem P*
- virtual Integer [getInfo](#) () const

### Static Public Member Functions

- static void [fcn](#) (int m, int n, double \*x, double \*fvec, int \*iflag)

### 7.516.2 Constructor & Destructor Documentation

7.516.2.1 [LevenbergMarquardt](#) (Real *epsfcn* = 1e-8, Real *ftol* = 1e-8, Real *xtol* = 1e-8, Real *gtol* = 1e-8, Size *maxfev* = 400)

Constructor taking as input the characteristic length and tolerance

## 7.517 LexicographicalView Class Template Reference

```
#include <ql/Math/lexicographicalview.hpp>
```

### 7.517.1 Detailed Description

**template<class RandomAccessIterator> class QuantLib::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 boost::reverse\_iterator< RandomAccessIterator > [reverse\\_x\\_iterator](#)  
*iterates backwards over  $v_{ij}$  with  $j$  fixed.*
- typedef [step\\_iterator](#)< RandomAccessIterator > [y\\_iterator](#)  
*iterates over  $v_{ij}$  with  $i$  fixed.*
- typedef boost::reverse\_iterator< [y\\_iterator](#) > [reverse\\_y\\_iterator](#)  
*iterates backwards over  $v_{ij}$  with  $i$  fixed.*

### Public Member Functions

- [LexicographicalView](#) (const RandomAccessIterator &begin, const RandomAccessIterator &end, Size xSize)  
*attaches the view with the given dimension to a sequence*

#### Element access

- [y\\_iterator](#) [operator\[\]](#) (Size i)

#### Iterator access

- [x\\_iterator](#) [xbegin](#) (Size j)
- [x\\_iterator](#) [xend](#) (Size j)
- [reverse\\_x\\_iterator](#) [rxbegin](#) (Size j)
- [reverse\\_x\\_iterator](#) [rxend](#) (Size j)
- [y\\_iterator](#) [ybegin](#) (Size i)
- [y\\_iterator](#) [yend](#) (Size i)
- [reverse\\_y\\_iterator](#) [rybegin](#) (Size i)
- [reverse\\_y\\_iterator](#) [ryend](#) (Size i)

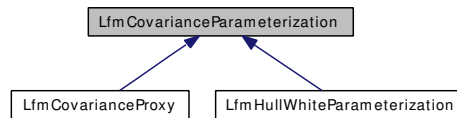
**Inspectors**

- Size `xSize ()` const  
*dimension of the array along  $x$*
- Size `ySize ()` const  
*dimension of the array along  $y$*

## 7.518 LfmCovarianceParameterization Class Reference

```
#include <ql/Processes/lfmcovarparam.hpp>
```

Inheritance diagram for LfmCovarianceParameterization:



### 7.518.1 Detailed Description

libor market model parameterization

Brigo, Damiano, Mercurio, Fabio, Morini, Massimo, 2003 Different Covariance Parameterizations of the [Libor](#) Market Model and Joint Caps/Swaptions Calibration (<http://www.exoticderivatives.com/Files/Papers/brigomercuriomorini.pdf>)

### Public Member Functions

- **LfmCovarianceParameterization** (Size size, Size factors)
- Size **size** () const
- Size **factors** () const
- virtual [Disposable](#)< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const=0
- virtual [Disposable](#)< [Matrix](#) > **covariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual [Disposable](#)< [Matrix](#) > **integratedCovariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const

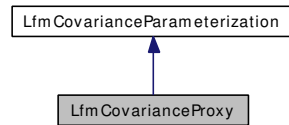
### Protected Attributes

- const Size **size\_**
- const Size **factors\_**

## 7.519 LfmCovarianceProxy Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lfmcovarproxy.hpp>
```

Inheritance diagram for LfmCovarianceProxy:



### 7.519.1 Detailed Description

proxy for a libor forward model covariance parameterization

#### Public Member Functions

- **LfmCovarianceProxy** (const boost::shared\_ptr< [LmVolatilityModel](#) > &volaModel, const boost::shared\_ptr< [LmCorrelationModel](#) > &corrModel)
- boost::shared\_ptr< [LmVolatilityModel](#) > **volatilityModel** () const
- boost::shared\_ptr< [LmCorrelationModel](#) > **correlationModel** () const
- [Disposable](#)< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **covariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual [Real](#) **integratedCovariance** (Size i, Size j, Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const

#### Protected Attributes

- const boost::shared\_ptr< [LmVolatilityModel](#) > **volaModel\_**
- const boost::shared\_ptr< [LmCorrelationModel](#) > **corrModel\_**

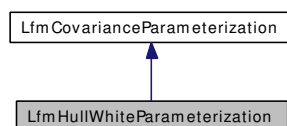
#### Friends

- class **Var\_Helper**

## 7.520 LfmHullWhiteParameterization Class Reference

#include <ql/Processes/lfmhullwhiteparam.hpp>

Inheritance diagram for LfmHullWhiteParameterization:



### 7.520.1 Detailed Description

libor market model parameterization based on Hull White paper

Hull, John, White, Alan, 1999, [Forward Rate Volatilities, Swap Rate Volatilities and the Implementation of the Libor Market Model](http://www.rotman.utoronto.ca/~amackay/fin/libormktmodel2.pdf) (<<http://www.rotman.utoronto.ca/~amackay/fin/libormktmodel2.pdf>>)

#### Tests

the correctness is tested by Monte-Carlo reproduction of caplet & ratchet npvs and comparison with Black pricing.

### Public Member Functions

- **LfmHullWhiteParameterization** (const boost::shared\_ptr< [LiborForwardModelProcess](#) > &process, const boost::shared\_ptr< [CapletVolatilityStructure](#) > &capletVol, const [Matrix](#) &correlation=[Matrix](#)(), Size factors=1)
- [Disposable](#)< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **covariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **integratedCovariance** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const

### Protected Member Functions

- Size **nextIndexReset** (Time t) const

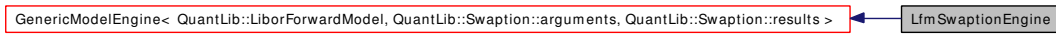
### Protected Attributes

- [Matrix](#) **diffusion\_**
- [Matrix](#) **covariance\_**
- std::vector< Time > **fixingTimes\_**

## 7.521 LfmSwaptionEngine Class Reference

```
#include <ql/PricingEngines/Swaption/lfmwaptionengine.hpp>
```

Inheritance diagram for LfmSwaptionEngine:



### 7.521.1 Detailed Description

libor forward model swaption engine based on black formula

#### Public Member Functions

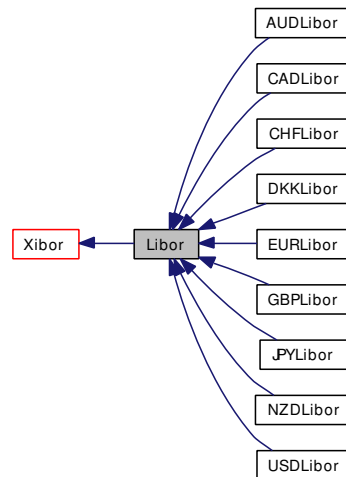
- **LfmSwaptionEngine** (const boost::shared\_ptr< [LiborForwardModel](#) > &model)
- void **calculate** () const



## 7.522 Libor Class Reference

```
#include <ql/Indexes/libor.hpp>
```

Inheritance diagram for Libor:



### 7.522.1 Detailed Description

base class for BBA LIBOR indexes

#### Public Member Functions

- **Libor** (const std::string &familyName, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Currency](#) &currency, const [Calendar](#) &localCalendar, const [Calendar](#) &currencyCalendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, const [Handle](#)< [YieldTermStructure](#) > &h)
- **Libor** (const std::string &familyName, const [Period](#) &tenor, [Integer](#) settlementDays, const [Currency](#) &currency, const [Calendar](#) &localCalendar, const [Calendar](#) &currencyCalendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, const [Handle](#)< [YieldTermStructure](#) > &h)

#### Date calculations

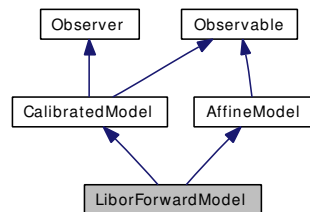
see <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1412>

- **Date** valueDate (const [Date](#) &fixingDate) const
- **Date** maturityDate (const [Date](#) &valueDate) const

## 7.523 LiborForwardModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp>
```

Inheritance diagram for LiborForwardModel:



### 7.523.1 Detailed Description

**Libor Forward** Model.

References:

Stefan Weber, 2005, Efficient Calibration for **Libor** Market Models, (<http://workshop.mathfinance.de/2005/papers/weber/slides.pdf>)

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of **Libor** Market Model and Joint Caps/Swaptions Calibration, ([http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo\\_D.pdf](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf))

#### Tests

the correctness is tested using Monte-Carlo Simulation to reproduce swaption npvs, model calibration and exact cap pricing

### Public Member Functions

- **LiborForwardModel** (const boost::shared\_ptr< **LiborForwardModelProcess** > &process, const boost::shared\_ptr< **LmVolatilityModel** > &volaModel, const boost::shared\_ptr< **LmCorrelationModel** > &corrModel)
- **Rate S\_0** (Size alpha, Size beta) const
- virtual boost::shared\_ptr< **SwaptionVolatilityMatrix** > **getSwaptionVolatilityMatrix** () const
- **DiscountFactor discount** (Time t) const  
*Implied discount curve.*
- Real **discountBond** (Time now, Time maturity, **Array** factors) const
- Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bond-Maturity) const
- void **setParams** (const **Array** &params)

### Protected Member Functions

- **Disposable**< **Array** > **w\_0** (Size alpha, Size beta) const

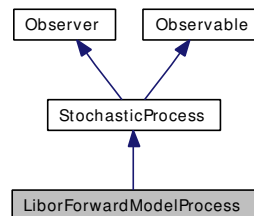
## Protected Attributes

- `std::vector< Real > f_`
- `std::vector< Time > accrualPeriod_`
- `const boost::shared_ptr< LfmCovarianceProxy > covarProxy_`
- `const boost::shared_ptr< LiborForwardModelProcess > process_`
- `boost::shared_ptr< SwaptionVolatilityMatrix > swaptionVola`

## 7.524 LiborForwardModelProcess Class Reference

```
#include <ql/Processes/lfmprocess.hpp>
```

Inheritance diagram for LiborForwardModelProcess:



### 7.524.1 Detailed Description

libor-forward-model process

stochastic process of a libor forward model using the rolling forward measure incl. predictor-corrector step

References:

Glasserman, Paul, 2004, Monte Carlo Methods in Financial Engineering, Springer, Section 3.7

Antoon Pelsser, 2000, Efficient Methods for Valuing Interest Rate Derivatives, Springer, 8

Hull, John, White, Alan, 1999, [Forward Rate Volatilities, Swap Rate Volatilities and the Implementation of the Libor Market Model](#) (<http://www.rotman.utoronto.ca/~amackay/fin/libormktmodel2.pdf>)

#### Tests

the correctness is tested by Monte-Carlo reproduction of caplet & ratchet NPVs and comparison with Black pricing.

#### Warning

this class does not work correctly with Visual C++ 6.

### Public Member Functions

- **LiborForwardModelProcess** (Size size, const boost::shared\_ptr< [Xibor](#) > &index)
- **Disposable**< [Array](#) > **initialValues** () const  
returns the initial values of the state variables
- **Disposable**< [Array](#) > **drift** (Time t, const [Array](#) &x) const  
returns the drift part of the equation, i.e.,  $\mu(t, x_t)$
- **Disposable**< [Matrix](#) > **diffusion** (Time t, const [Array](#) &x) const  
returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$
- **Disposable**< [Matrix](#) > **covariance** (Time t0, const [Array](#) &x0, Time dt) const

- `Disposable< Array > apply (const Array &x0, const Array &dx) const`
- `Disposable< Array > evolve (Time t0, const Array &x0, Time dt, const Array &dw) const`
- `Size size () const`  
*returns the number of dimensions of the stochastic process*
- `Size factors () const`  
*returns the number of independent factors of the process*
- `boost::shared_ptr< Xibor > index () const`
- `std::vector< boost::shared_ptr< CashFlow > > cashFlows (Real amount=1.0) const`
- `void setCovarParam (const boost::shared_ptr< LfmCovarianceParameterization > &param)`
- `boost::shared_ptr< LfmCovarianceParameterization > covarParam () const`
- `Size nextIndexReset (Time t) const`
- `const std::vector< Time > & fixingTimes () const`
- `const std::vector< Date > & fixingDates () const`
- `const std::vector< Time > & accrualStartTimes () const`
- `const std::vector< Time > & accrualEndTimes () const`
- `std::vector< DiscountFactor > discountBond (const std::vector< Rate > &rates) const`

## 7.524.2 Member Function Documentation

### 7.524.2.1 `Disposable<Matrix> covariance (Time t0, const Array &x0, Time dt) const` [virtual]

returns the covariance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

### 7.524.2.2 `Disposable<Array> apply (const Array &x0, const Array &dx) const` [virtual]

applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented from [StochasticProcess](#).

### 7.524.2.3 `Disposable<Array> evolve (Time t0, const Array &x0, Time dt, const Array &dw) const` [virtual]

returns the asset value after a time interval  $\Delta t$  according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where  $E$  is the expectation and  $S$  the standard deviation.

Reimplemented from [StochasticProcess](#).

## 7.525 Linear Class Reference

```
#include <ql/Math/linearinterpolation.hpp>
```

### 7.525.1 Detailed Description

[Linear](#) interpolation factory and traits.

#### Public Types

- enum { **global** = 0 }

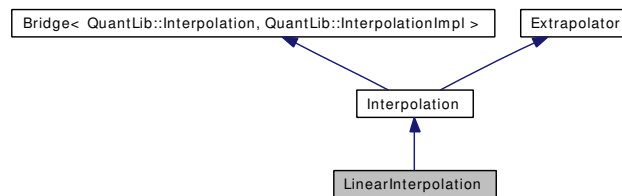
#### Public Member Functions

- template<class I1, class I2> [Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

## 7.526 LinearInterpolation Class Reference

```
#include <ql/Math/linearinterpolation.hpp>
```

Inheritance diagram for LinearInterpolation:



### 7.526.1 Detailed Description

Linear interpolation between discrete points

#### Public Member Functions

- `template<class I1, class I2> LinearInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)`

### 7.526.2 Constructor & Destructor Documentation

#### 7.526.2.1 [LinearInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

**Precondition:**

the *x* values must be sorted.

## 7.527 LinearLeastSquaresRegression Class Template Reference

```
#include <ql/Math/linearleastsquaresregression.hpp>
```

### 7.527.1 Detailed Description

```
template<class ArgumentType = Real> class QuantLib::LinearLeastSquaresRegression<
ArgumentType >
```

general linear least squares regression

References: "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

#### Tests

the correctness of the returned values is tested by checking their properties.

### Public Member Functions

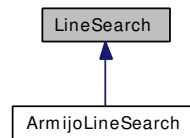
- **LinearLeastSquaresRegression** (const std::vector< ArgumentType > &x, const std::vector< Real > &y, const std::vector< boost::function1< Real, ArgumentType > > &v)
- const [Array](#) & **a** () const
- const [Array](#) & **err** () const



## 7.528 LineSearch Class Reference

```
#include <ql/Optimization/linesearch.hpp>
```

Inheritance diagram for LineSearch:



### 7.528.1 Detailed Description

Base class for line search.

#### Public Member Functions

- [LineSearch](#) (Real=0.0)  
*Default constructor.*
- virtual [~LineSearch](#) ()  
*Destructor.*
- const [Array](#) & [lastX](#) ()  
*return last x value*
- Real [lastFunctionValue](#) ()  
*return last cost function value*
- const [Array](#) & [lastGradient](#) ()  
*return last gradient*
- Real [lastGradientNorm2](#) ()  
*return square norm of last gradient*
- bool [succeed](#) ()
- virtual Real [operator\(\)](#) (const [Problem](#) &P, Real t\_ini)=0  
*Perform line search.*
- Real [update](#) ([Array](#) &params, const [Array](#) &direction, Real beta, const [Constraint](#) &constraint)

#### Protected Attributes

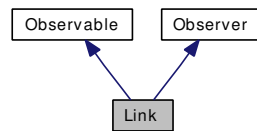
- [Array](#) [xtd\\_](#)  
*new x and its gradient*

- [Array](#) **gradient\_**
- Real [qt\\_](#)  
*cost function value and gradient norm corresponding to xtd\_*
- Real **qpt\_**
- bool [succeed\\_](#)  
*flag to know if linesearch succeed*

## 7.529 Link Class Template Reference

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

Inheritance diagram for Link:



### 7.529.1 Detailed Description

```
template<class T> class QuantLib::Link< T >
```

Relinkable access to a shared pointer.

#### Precondition:

Class T must inherit from [Observable](#)

#### Public Member Functions

- [Link](#) (const boost::shared\_ptr< T > &h=boost::shared\_ptr< T >(), bool registerAsObserver=true)
- void [linkTo](#) (const boost::shared\_ptr< T > &, bool registerAsObserver=true)
- bool [empty](#) () const  
*Checks if the contained shared pointer points to anything.*
- const boost::shared\_ptr< T > & [currentLink](#) () const  
*Returns the contained shared pointer.*
- void [swap](#) ([Link](#)< T > &other)  
*Swaps two links.*
- void [update](#) ()  
*[Observer](#) interface.*

#### Related Functions

(Note that these are not member functions.)

- void [swap](#) ([Link](#)< T > &, [Link](#)< T > &)

## 7.529.2 Constructor & Destructor Documentation

7.529.2.1 [Link](#) (const boost::shared\_ptr< T > & *h* = boost::shared\_ptr< T >(), bool *registerAsObserver* = true) [explicit]

### Warning

see the documentation of the [linkTo\(\)](#) method for issues relatives to `registerAsObserver`.

## 7.529.3 Member Function Documentation

7.529.3.1 void `linkTo` (const boost::shared\_ptr< T > &, bool *registerAsObserver* = true)

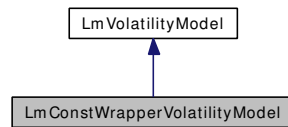
### Warning

`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 shared pointer 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 handle to register as observer of such a shared pointer, it is his responsibility to ensure that the handle gets destroyed before the pointed object does.

## 7.530 LmConstWrapperVolatilityModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmconstwrappervolmodel.hpp>
```

Inheritance diagram for LmConstWrapperVolatilityModel:



### 7.530.1 Detailed Description

caplet const volatility model

#### Public Member Functions

- **LmConstWrapperVolatilityModel** (const boost::shared\_ptr< [LmVolatilityModel](#) > &volaModel)
- [Disposable](#)< [Array](#) > **volatility** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Volatility](#) **volatility** (Size i, Time t, const [Array](#) &x=[Null](#)< [Array](#) >())
- [Real integrated Variance](#) (Size i, Size j, Time u, const [Array](#) &x=[Null](#)< [Array](#) >()) const

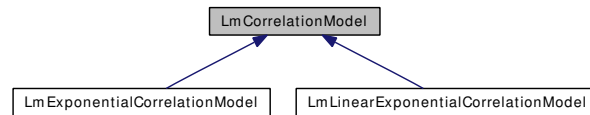
#### Protected Attributes

- const boost::shared\_ptr< [LmVolatilityModel](#) > **volaModel\_**

## 7.531 LmCorrelationModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp>
```

Inheritance diagram for LmCorrelationModel:



### 7.531.1 Detailed Description

libor forward correlation model

#### Public Member Functions

- **LmCorrelationModel** (Size size, Size nArguments)
- virtual Size **size** () const
- virtual Size **factors** () const
- std::vector< [Parameter](#) > & **params** ()
- void **setParams** (const std::vector< [Parameter](#) > &arguments)
- virtual [Disposable](#)< [Matrix](#) > **correlation** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const=0
- virtual [Disposable](#)< [Matrix](#) > **pseudoSqrt** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual [Real](#) **correlation** (Size i, Size j, Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual bool **isTimeIndependent** () const

#### Protected Member Functions

- virtual void **generateArguments** ()=0

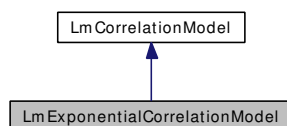
#### Protected Attributes

- const Size **size\_**
- std::vector< [Parameter](#) > **arguments\_**

## 7.532 LmExponentialCorrelationModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmexpcorrmodel.hpp>
```

Inheritance diagram for LmExponentialCorrelationModel:



### 7.532.1 Detailed Description

exponential correlation model

This class describes a exponential correlation model

$$\rho_{i,j} = e^{(-\beta\|i-j\|)}$$

References:

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of [Libor](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf) Market Model and Joint Caps/Swaptions Calibration, (<[http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo\\_D.pdf](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf)>)

### Public Member Functions

- **LmExponentialCorrelationModel** (Size size, [Real](#) rho)
- [Disposable](#)< [Matrix](#) > **correlation** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **pseudoSqrt** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Real](#) **correlation** (Size i, Size j, Time t, const [Array](#) &x) const
- bool **isTimeIndependent** () const

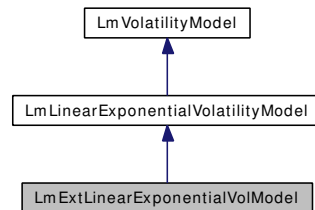
### Protected Member Functions

- void **generateArguments** ()

## 7.533 LmExtLinearExponentialVolModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmextlinexpvolmodel.hpp>
```

Inheritance diagram for LmExtLinearExponentialVolModel:



### 7.533.1 Detailed Description

extended linear exponential volatility model

This class describes an extended linear-exponential volatility model

$$\sigma_i(t) = k_i * ((a * (T_i - t) + d) * e^{-b(T_i - t)} + c)$$

References:

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of [Libor](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf) Market Model and Joint Caps/Swaptions Calibration, (<[http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo\\_D.pdf](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf)>)

### Public Member Functions

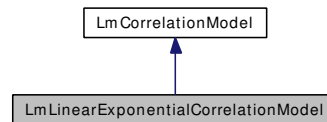
- **LmExtLinearExponentialVolModel** (const std::vector< Time > &fixingTimes, [Real](#) a, [Real](#) b, [Real](#) c, [Real](#) d)
- [Disposable](#)< [Array](#) > **volatility** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Volatility](#) **volatility** (Size i, Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Real](#) **integratedVariance** (Size i, Size j, Time u, const [Array](#) &x=[Null](#)< [Array](#) >()) const



## 7.534 LmLinearExponentialCorrelationModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmlinexpcorrmodel.hpp>
```

Inheritance diagram for LmLinearExponentialCorrelationModel:



### 7.534.1 Detailed Description

linear exponential correlation model

This class describes a exponential correlation model

$$\rho_{i,j} = rho + (1 - rho) * e^{(-\beta||i-j||)}$$

References:

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of [Libor](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf) Market Model and Joint Caps/Swaptions Calibration, (<[http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo\\_D.pdf](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf)>)

### Public Member Functions

- **LmLinearExponentialCorrelationModel** (Size size, [Real](#) rho, [Real](#) beta)
- [Disposable](#)< [Matrix](#) > **correlation** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Disposable](#)< [Matrix](#) > **pseudoSqrt** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- [Real](#) **correlation** (Size i, Size j, Time t, const [Array](#) &x) const
- bool **isTimeIndependent** () const

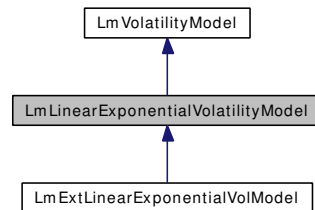
### Protected Member Functions

- void **generateArguments** ()

## 7.535 LmLinearExponentialVolatilityModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmlinepvolmodel.hpp>
```

Inheritance diagram for LmLinearExponentialVolatilityModel:



### 7.535.1 Detailed Description

linear exponential volatility model

This class describes a linear-exponential volatility model

$$\sigma_i(t) = (a * (T_i - t) + d) * e^{-b(T_i - t)} + c$$

References:

Damiano Brigo, Fabio Mercurio, Massimo Morini, 2003, Different Covariance Parameterizations of Libor Market Model and Joint Caps/Swaptions Calibration, (<[http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo\\_D.pdf](http://www.business.uts.edu.au/qfrc/conferences/qmf2001/Brigo_D.pdf)>)

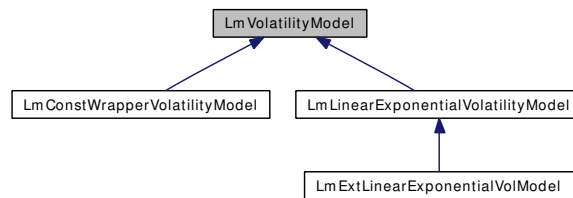
### Public Member Functions

- **LmLinearExponentialVolatilityModel** (const std::vector< Time > &fixingTimes, Real a, Real b, Real c, Real d)
- **Disposable< Array > volatility** (Time t, const Array &x=NULL< Array >()) const
- **Volatility volatility** (Size i, Time t, const Array &x=NULL< Array >()) const
- **Real integratedVariance** (Size i, Size j, Time u, const Array &x=NULL< Array >()) const

## 7.536 LmVolatilityModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp>
```

Inheritance diagram for LmVolatilityModel:



### 7.536.1 Detailed Description

caplet volatility model

#### Public Member Functions

- **LmVolatilityModel** (Size size, Size nArguments)
- Size **size** () const
- std::vector< [Parameter](#) > & **params** ()
- void **setParams** (const std::vector< [Parameter](#) > &arguments)
- virtual [Disposable](#)< [Array](#) > **volatility** (Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const=0
- virtual [Volatility](#) **volatility** (Size i, Time t, const [Array](#) &x=[Null](#)< [Array](#) >()) const
- virtual [Real](#) **integratedVariance** (Size i, Size j, Time u, const [Array](#) &x=[Null](#)< [Array](#) >()) const

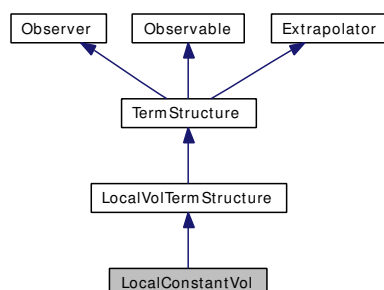
#### Protected Attributes

- const Size **size\_**
- std::vector< [Parameter](#) > **arguments\_**

## 7.537 LocalConstantVol Class Reference

```
#include <ql/Volatilities/localconstantvol.hpp>
```

Inheritance diagram for LocalConstantVol:



### 7.537.1 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](#).

### Public Member Functions

- **LocalConstantVol** (const [Date](#) &referenceDate, Volatility volatility, const [DayCounter](#) &dayCounter)
- **LocalConstantVol** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- **LocalConstantVol** (Integer settlementDays, const [Calendar](#) &, Volatility volatility, const [DayCounter](#) &dayCounter)
- **LocalConstantVol** (Integer settlementDays, const [Calendar](#) &, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)

### LocalVolTermStructure interface

- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*
- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- Time **maxTime** () const  
*the latest time for which the curve can return values*
- Real **minStrike** () const  
*the minimum strike for which the term structure can return vols*
- Real **maxStrike** () const  
*the maximum strike for which the term structure can return vols*

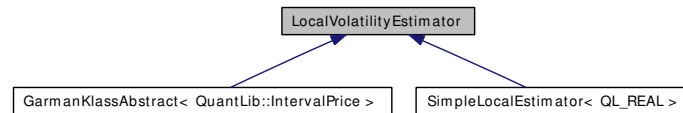
**Visitability**

- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.538 LocalVolatilityEstimator Class Template Reference

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

Inheritance diagram for LocalVolatilityEstimator:



### 7.538.1 Detailed Description

```
template<class T> class QuantLib::LocalVolatilityEstimator< T >
```

The volatility model classes are of two types. The first

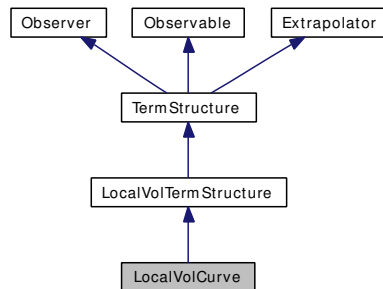
#### Public Member Functions

- virtual [TimeSeries](#)< [Volatility](#) > **calculate** (const [TimeSeries](#)< T > &quoteSeries)=0

## 7.539 LocalVolCurve Class Reference

#include <ql/Volatilities/localvolcurve.hpp>

Inheritance diagram for LocalVolCurve:



### 7.539.1 Detailed Description

Local volatility curve derived from a Black curve.

#### Public Member Functions

- **LocalVolCurve** (const [Handle](#)< [BlackVarianceCurve](#) > &curve)

#### LocalVolTermStructure interface

- const [Date](#) & [referenceDate](#) () const  
*the date at which discount = 1.0 and/or variance = 0.0*
- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return values*
- Real [minStrike](#) () const  
*the minimum strike for which the term structure can return vols*
- Real [maxStrike](#) () const  
*the maximum strike for which the term structure can return vols*

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

#### Protected Member Functions

- [Volatility](#) [localVolImpl](#) (Time, Real) const

## 7.539.2 Member Function Documentation

### 7.539.2.1 [Volatility](#) localVolImpl (Time $t$ , Real *dummy*) 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

$$\sigma_L(t) = \sqrt{\frac{d}{dt} \sigma_B^2(t) t}$$

can be deduced which is here implemented.

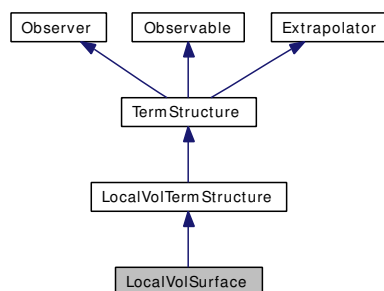
Implements [LocalVolTermStructure](#).



## 7.540 LocalVolSurface Class Reference

```
#include <ql/Volatilities/localvolsurface.hpp>
```

Inheritance diagram for LocalVolSurface:



### 7.540.1 Detailed Description

Local volatility surface derived from a Black vol surface.

For details about this implementation refer to "Stochastic Volatility and Local Volatility," in "Case Studies and Financial Modelling Course Notes," by Jim Gatheral, Fall Term, 2003

see [www.math.nyu.edu/fellows\\_fin\\_math/gatheral/Lecture1\\_Fall02.pdf](http://www.math.nyu.edu/fellows_fin_math/gatheral/Lecture1_Fall02.pdf)

#### Bug

this class is untested, probably unreliable.

### Public Member Functions

- **LocalVolSurface** (const [Handle](#)< [BlackVolTermStructure](#) > &blackTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > &dividendTS, const [Handle](#)< [Quote](#) > &underlying)
- **LocalVolSurface** (const [Handle](#)< [BlackVolTermStructure](#) > &blackTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > &dividendTS, Real underlying)

#### LocalVolTermStructure interface

- const [Date](#) & [referenceDate](#) () const  
*the date at which discount = 1.0 and/or variance = 0.0*
- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return values*
- Real [minStrike](#) () const  
*the minimum strike for which the term structure can return vols*

- Real [maxStrike](#) () const  
*the maximum strike for which the term structure can return vols*

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

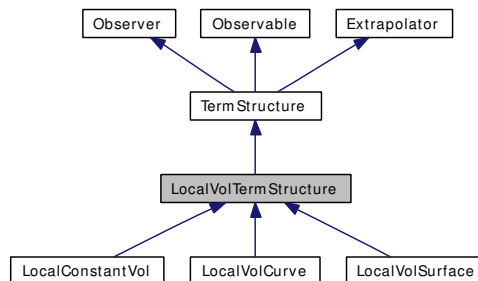
#### Protected Member Functions

- [Volatility localVolImpl](#) (Time, Real) const  
*local vol calculation*

## 7.541 LocalVolTermStructure Class Reference

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

Inheritance diagram for LocalVolTermStructure:



### 7.541.1 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

- **Volatility** **localVol** (const **Date** &d, Real underlyingLevel, bool extrapolate=false) const
- **Volatility** **localVol** (Time t, Real underlyingLevel, bool extrapolate=false) const

#### Limits

- virtual Real **minStrike** () const=0  
*the minimum strike for which the term structure can return vols*
- virtual Real **maxStrike** () const=0  
*the maximum strike for which the term structure can return vols*

#### Visitability

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

### Protected Member Functions

#### Calculations

*These methods must be implemented in derived classes to perform the actual volatility calculations. When they are called, range check has already been performed; therefore, they must assume that extrapolation is required.*

- virtual [Volatility](#) [localVolImpl](#) (Time t, Real strike) const =0  
*local vol calculation*

## 7.541.2 Constructor & Destructor Documentation

### 7.541.2.1 [LocalVolTermStructure](#) ()

default constructor

#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

### 7.541.2.2 [LocalVolTermStructure](#) ()

default constructor

#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.542 LogLinear Class Reference

```
#include <ql/Math/loglinearinterpolation.hpp>
```

### 7.542.1 Detailed Description

log-linear interpolation factory and traits

#### Public Types

- enum { **global** = 0 }

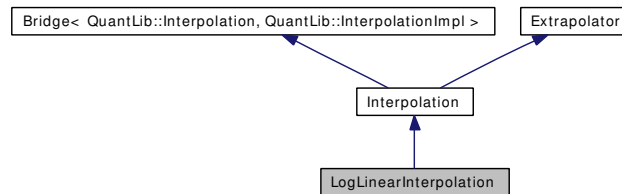
#### Public Member Functions

- template<class I1, class I2> [Interpolation](#) **interpolate** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin) const

## 7.543 LogLinearInterpolation Class Reference

```
#include <ql/Math/loglinearinterpolation.hpp>
```

Inheritance diagram for LogLinearInterpolation:



### 7.543.1 Detailed Description

log-linear interpolation between discrete points

#### Todo

implement primitive, derivative, and secondDerivative functions.

### Public Member Functions

- `template<class I1, class I2> LogLinearInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)`

### 7.543.2 Constructor & Destructor Documentation

7.543.2.1 [LogLinearInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

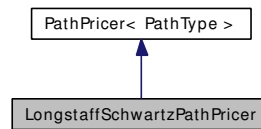
#### Precondition:

the  $x$  values must be sorted.

## 7.544 LongstaffSchwartzPathPricer Class Template Reference

```
#include <ql/MonteCarlo/longstaffschwartzpathpricer.hpp>
```

Inheritance diagram for LongstaffSchwartzPathPricer:



### 7.544.1 Detailed Description

```
template<class PathType> class QuantLib::LongstaffSchwartzPathPricer< PathType >
```

Longstaff-Schwarz path pricer for early exercise options.

References:

Francis Longstaff, Eduardo Schwartz, 2001. Valuing American Options by Simulation: A Simple Least-Squares Approach, The Review of Financial Studies, Volume 14, No. 1, 113-147

#### Tests

the correctness of the returned value is tested by reproducing results available in web/literature

### Public Types

- typedef EarlyExerciseTraits< PathType >::StateType **StateType**

### Public Member Functions

- **LongstaffSchwartzPathPricer** (const [TimeGrid](#) &times, const boost::shared\_ptr< [EarlyExercisePathPricer](#)< PathType > &, const boost::shared\_ptr< [YieldTermStructure](#) > &termStructure)
- Real **operator()** (const PathType &path) const
- virtual void **calibrate** ()

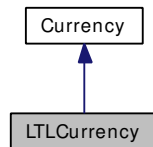
### Protected Attributes

- bool **calibrationPhase\_**
- const boost::shared\_ptr< [EarlyExercisePathPricer](#)< PathType > > **pathPricer\_**
- boost::scoped\_array< [Array](#) > **coeff\_**
- boost::scoped\_array< DiscountFactor > **dF\_**
- std::vector< PathType > **paths\_**
- const std::vector< boost::function1< Real, StateType > > **v\_**

## 7.545 LTLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for LTLCurrency:



### 7.545.1 Detailed Description

Lithuanian litas.

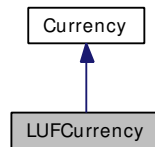
The ISO three-letter code is `LTL`; the numeric code is 440. It is divided in 100 centu.



## 7.546 LUFCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for LUFCurrency:



### 7.546.1 Detailed Description

Luxembourg franc.

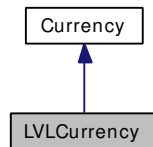
The ISO three-letter code was LUF; the numeric code was 442. It was divided in 100 centimes.

Obsoleted by the Euro since 1999.

## 7.547 LVLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for LVLCurrency:



### 7.547.1 Detailed Description

Latvian lat.

The ISO three-letter code is LVL; the numeric code is 428. It is divided in 100 santims.

## 7.548 MakeMCAmericanEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mcamericanengine.hpp>
```

### 7.548.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::Make-  
MCAmericanEngine< RNG, S >
```

Monte Carlo American engine factory.

Examples:

[EquityOption.cpp](#).

### Public Member Functions

- [MakeMCAmericanEngine](#) & **withSteps** (Size steps)
- [MakeMCAmericanEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCAmericanEngine](#) & **withSamples** (Size samples)
- [MakeMCAmericanEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCAmericanEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCAmericanEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCAmericanEngine](#) & **withAntitheticVariate** (bool b=true)
- [MakeMCAmericanEngine](#) & **withControlVariate** (bool b=true)
- [MakeMCAmericanEngine](#) & **withPolynomOrder** (Size polynomOrder)
- [MakeMCAmericanEngine](#) & **withBasisSystem** (LsmBasisSystem::PolynomType)
- [MakeMCAmericanEngine](#) & **withCalibrationSamples** (Size calibrationSamples)
- **operator boost::shared\_ptr () const**

## 7.549 MakeMCDigitalEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mcdigitalengine.hpp>
```

### 7.549.1 Detailed Description

`template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MakeMCDigitalEngine< RNG, S >`

Monte Carlo digital engine factory.

### Public Member Functions

- [MakeMCDigitalEngine](#) & **withSteps** (Size steps)
- [MakeMCDigitalEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCDigitalEngine](#) & **withBrownianBridge** (bool b=true)
- [MakeMCDigitalEngine](#) & **withSamples** (Size samples)
- [MakeMCDigitalEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCDigitalEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCDigitalEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCDigitalEngine](#) & **withAntitheticVariate** (bool b=true)
- [MakeMCDigitalEngine](#) & **withControlVariate** (bool b=true)
- **operator boost::shared\_ptr () const**

## 7.550 MakeMCEuropeanEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mceuropeanengine.hpp>
```

### 7.550.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::Make-  
MCEuropeanEngine< RNG, S >
```

Monte Carlo European engine factory.

Examples:

[EquityOption.cpp](#).

### Public Member Functions

- [MakeMCEuropeanEngine](#) & **withSteps** (Size steps)
- [MakeMCEuropeanEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCEuropeanEngine](#) & **withBrownianBridge** (bool b=true)
- [MakeMCEuropeanEngine](#) & **withSamples** (Size samples)
- [MakeMCEuropeanEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCEuropeanEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCEuropeanEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCEuropeanEngine](#) & **withAntitheticVariate** (bool b=true)
- [MakeMCEuropeanEngine](#) & **withControlVariate** (bool b=true)
- **operator boost::shared\_ptr ()** const

## 7.551 MakeMCEuropeanHestonEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp>
```

### 7.551.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::Make-  
MCEuropeanHestonEngine< RNG, S >
```

Monte Carlo Heston European engine factory.

### Public Member Functions

- [MakeMCEuropeanHestonEngine](#) & **withSteps** (Size steps)
- [MakeMCEuropeanHestonEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCEuropeanHestonEngine](#) & **withSamples** (Size samples)
- [MakeMCEuropeanHestonEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCEuropeanHestonEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCEuropeanHestonEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCEuropeanHestonEngine](#) & **withAntitheticVariate** (bool b=true)
- **operator boost::shared\_ptr** () const

## 7.552 MakeMCHullWhiteCapFloorEngine Class Template Reference

```
#include <ql/PricingEngines/CapFloor/mchullwhiteengine.hpp>
```

### 7.552.1 Detailed Description

`template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MakeMCHullWhiteCapFloorEngine< RNG, S >`

Monte Carlo Hull-White cap-floor engine factory.

### Public Member Functions

- `MakeMCHullWhiteCapFloorEngine` (const boost::shared\_ptr< [HullWhite](#) > &)
- `MakeMCHullWhiteCapFloorEngine` & `withBrownianBridge` (bool b=true)
- `MakeMCHullWhiteCapFloorEngine` & `withSamples` (Size samples)
- `MakeMCHullWhiteCapFloorEngine` & `withTolerance` (Real tolerance)
- `MakeMCHullWhiteCapFloorEngine` & `withMaxSamples` (Size samples)
- `MakeMCHullWhiteCapFloorEngine` & `withSeed` (BigNatural seed)
- `MakeMCHullWhiteCapFloorEngine` & `withAntitheticVariate` (bool b=true)
- `operator boost::shared_ptr () const`

## 7.553 MakeMCVarianceSwapEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/mcvarianceswapengine.hpp>
```

### 7.553.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::Make-  
MCVarianceSwapEngine< RNG, S >
```

Monte Carlo variance-swap engine factory.

### Public Member Functions

- [MakeMCVarianceSwapEngine](#) & **withSteps** (Size steps)
- [MakeMCVarianceSwapEngine](#) & **withStepsPerYear** (Size steps)
- [MakeMCVarianceSwapEngine](#) & **withBrownianBridge** (bool b=true)
- [MakeMCVarianceSwapEngine](#) & **withSamples** (Size samples)
- [MakeMCVarianceSwapEngine](#) & **withTolerance** (Real tolerance)
- [MakeMCVarianceSwapEngine](#) & **withMaxSamples** (Size samples)
- [MakeMCVarianceSwapEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCVarianceSwapEngine](#) & **withAntitheticVariate** (bool b=true)
- **operator boost::shared\_ptr** () const



## 7.554 MakeSchedule Class Reference

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

### 7.554.1 Detailed Description

helper class

This class provides a more comfortable interface to the argument list of Schedule's constructor.

### Public Member Functions

- **MakeSchedule** (const [Date](#) &effectiveDate, const [Date](#) &terminationDate, const [Period](#) &tenor, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention)
- **MakeSchedule** (const [Calendar](#) &calendar, const [Date](#) &startDate, const [Date](#) &endDate, [Frequency](#) frequency, [BusinessDayConvention](#) convention)
- **MakeSchedule** & **withStubDate** (const [Date](#) &d)
- **MakeSchedule** & **longFinalPeriod** (bool flag=true)
- **MakeSchedule** & **shortFinalPeriod** (bool flag=true)
- **MakeSchedule** & **terminationDateConvention** ([BusinessDayConvention](#) conv)
- **MakeSchedule** & **backwards** (bool flag=true)
- **MakeSchedule** & **forwards** (bool flag=true)
- **MakeSchedule** & **endOfMonth** (bool flag=true)
- **MakeSchedule** & **withFirstDate** (const [Date](#) &d)
- **MakeSchedule** & **withNextToLastDate** (const [Date](#) &d)
- **operator Schedule** () const

## 7.555 MakeVanillaSwap Class Reference

```
#include <ql/Instruments/vanillaswap.hpp>
```

### 7.555.1 Detailed Description

helper class

This class provides a more comfortable way to instantiate standard market swap.

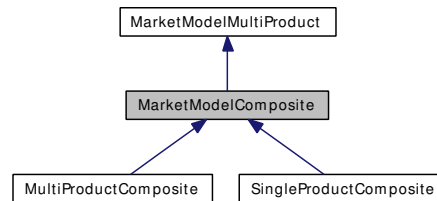
### Public Member Functions

- **MakeVanillaSwap** (const [Date](#) &effectiveDate, const [Period](#) &swapTenor, const [Calendar](#) &cal, Rate fixedRate, const boost::shared\_ptr< [Xibor](#) > &index, const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- **MakeVanillaSwap** & **receiveFixed** (bool flag=true)
- **MakeVanillaSwap** & **withNominal** (Real n)
- **MakeVanillaSwap** & **withFixedLegTenor** (const [Period](#) &t)
- **MakeVanillaSwap** & **withFixedLegCalendar** (const [Calendar](#) &cal)
- **MakeVanillaSwap** & **withFixedLegConvention** ([BusinessDayConvention](#) bdc)
- **MakeVanillaSwap** & **withFixedLegTerminationDateConvention** ([BusinessDayConvention](#) bdc)
- **MakeVanillaSwap** & **withFixedLegForward** (bool flag=true)
- **MakeVanillaSwap** & **withFixedLegNotEndOfMonth** (bool flag=true)
- **MakeVanillaSwap** & **withFixedLegFirstDate** (const [Date](#) &d)
- **MakeVanillaSwap** & **withFixedLegNextToLastDate** (const [Date](#) &d)
- **MakeVanillaSwap** & **withFixedLegDayCount** (const [DayCounter](#) &dc)
- **MakeVanillaSwap** & **withFloatingLegTenor** (const [Period](#) &t)
- **MakeVanillaSwap** & **withFloatingLegCalendar** (const [Calendar](#) &cal)
- **MakeVanillaSwap** & **withFloatingLegConvention** (const [BusinessDayConvention](#) bdc)
- **MakeVanillaSwap** & **withFloatingLegTerminationDateConvention** ([BusinessDayConvention](#) bdc)
- **MakeVanillaSwap** & **withFloatingLegForward** (bool flag=true)
- **MakeVanillaSwap** & **withFloatingLegNotEndOfMonth** (bool flag=true)
- **MakeVanillaSwap** & **withFloatingLegFirstDate** (const [Date](#) &d)
- **MakeVanillaSwap** & **withFloatingLegNextToLastDate** (const [Date](#) &d)
- **MakeVanillaSwap** & **withFloatingLegDayCount** (const [DayCounter](#) &dc)
- **MakeVanillaSwap** & **withFloatingLegSpread** (Spread sp)
- **operator VanillaSwap** () const
- **operator boost::shared\_ptr** () const

## 7.556 MarketModelComposite Class Reference

```
#include <ql/MarketModels/Products/compositeproduct.hpp>
```

Inheritance diagram for MarketModelComposite:



### 7.556.1 Detailed Description

Composition of two or more market-model products.

Instances of this class build a market-model product by composing one or more subproducts.

#### Precondition:

All subproducts must have the same rate times.

### Public Member Functions

#### MarketModelMultiProduct interface

- const [EvolutionDescription](#) & **evolution** () const
- std::vector< Size > **suggestedNumeraires** () const
- std::vector< Time > **possibleCashFlowTimes** () const
- void [reset](#) ()

*during simulation put product at start of path*

#### Composite facilities

- void **add** (const [Clone](#)< [MarketModelMultiProduct](#) > &, [Real](#) multiplier=1.0)
- void **subtract** (const [Clone](#)< [MarketModelMultiProduct](#) > &, [Real](#) multiplier=1.0)
- void **finalize** ()

### Protected Types

- typedef std::vector< SubProduct >::iterator **iterator**
- typedef std::vector< SubProduct >::const\_iterator **const\_iterator**

### Protected Attributes

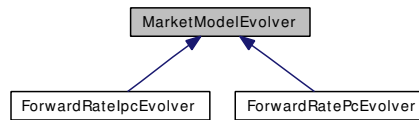
- std::vector< SubProduct > **components\_**
- std::vector< Time > **rateTimes\_**
- std::vector< Time > **evolutionTimes\_**

- [EvolutionDescription](#) **evolution\_**
- bool **finalized\_**
- Size **currentIndex\_**
- std::vector< Time > **cashflowTimes\_**
- std::vector< std::vector< Time > > **allEvolutionTimes\_**
- std::vector< std::vector< bool > > **isInSubset\_**

## 7.557 MarketModelEvolver Class Reference

```
#include <ql/MarketModels/marketmodelevolver.hpp>
```

Inheritance diagram for MarketModelEvolver:



### 7.557.1 Detailed Description

Abstract base class. The evolver does the actual gritty work of evolving the forward rates from one time to the next.

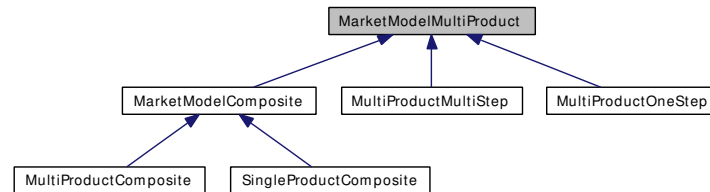
#### Public Member Functions

- virtual const std::vector< Size > & **numeraires** () const=0
- virtual [Real](#) **startNewPath** ()=0
- virtual [Real](#) **advanceStep** ()=0
- virtual Size **currentStep** () const=0
- virtual const [CurveState](#) & **currentState** () const=0

## 7.558 MarketModelMultiProduct Class Reference

```
#include <ql/MarketModels/marketmodelproduct.hpp>
```

Inheritance diagram for MarketModelMultiProduct:



### 7.558.1 Detailed Description

This is the abstract base class that encapsulates the notion of a product: it contains the information that would be in the termsheet of the product.

It's useful to have it be able to do several products simultaneously. The products would have to have the same underlying rate times of course. The class is therefore really encapsulating the notion of a multi-product.

For each time evolved to, it generates the cash flows associated to that time for the state of the yield curve. If one was doing a callable product then this would encompass the product and its exercise strategy.

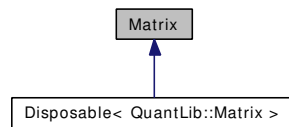
### Public Member Functions

- virtual std::vector< Size > **suggestedNumeraires** () const=0
- virtual const [EvolutionDescription](#) & **evolution** () const=0
- virtual std::vector< [Time](#) > **possibleCashFlowTimes** () const=0
- virtual Size **numberOfProducts** () const=0
- virtual Size **maxNumberOfCashFlowsPerProductPerStep** () const=0
- virtual void [reset](#) ()=0  
*during simulation put product at start of path*
- virtual bool [nextTimeStep](#) (const [CurveState](#) &currentState, std::vector< Size > &numberCashFlowsThisStep, std::vector< std::vector< CashFlow > > &cashFlowsGenerated)=0  
*return value indicates whether path is finished, TRUE means done*
- virtual std::auto\_ptr< [MarketModelMultiProduct](#) > [clone](#) () const=0  
*returns a newly-allocated copy of itself*

## 7.559 Matrix Class Reference

```
#include <ql/Math/matrix.hpp>
```

Inheritance diagram for Matrix:



### 7.559.1 Detailed Description

Matrix used in linear algebra.

This class implements the concept of [Matrix](#) as used in linear algebra. As such, it is **not** meant to be used as a container.

### 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, Real 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](#) > &)

### Public Types

- typedef Real \* **iterator**
- typedef const Real \* **const\_iterator**
- typedef boost::reverse\_iterator< iterator > **reverse\_iterator**
- typedef boost::reverse\_iterator< const\_iterator > **const\_reverse\_iterator**
- typedef Real \* **row\_iterator**
- typedef const Real \* **const\_row\_iterator**
- typedef boost::reverse\_iterator< row\_iterator > **reverse\_row\_iterator**
- typedef boost::reverse\_iterator< const\_row\_iterator > **const\_reverse\_row\_iterator**
- typedef [step\\_iterator](#)< iterator > **column\_iterator**
- typedef [step\\_iterator](#)< const\_iterator > **const\_column\_iterator**
- typedef boost::reverse\_iterator< [column\\_iterator](#) > **reverse\_column\_iterator**
- typedef boost::reverse\_iterator< [const\\_column\\_iterator](#) > **const\_reverse\_column\_iterator**

## Public Member Functions

### Algebraic operators

- const [Matrix](#) & **operator+=** (const [Matrix](#) &)
- const [Matrix](#) & **operator-=** (const [Matrix](#) &)
- const [Matrix](#) & **operator \*=** (Real)
- const [Matrix](#) & **operator/=** (Real)

### 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
- const\_row\_iterator **at** (Size) const
- row\_iterator **operator[]** (Size)
- row\_iterator **at** (Size)
- [Disposable](#)< [Array](#) > **diagonal** (void) const

### Inspectors

- Size **rows** () const
- Size **columns** () const
- bool **empty** () const

### Utilities

- void **swap** ([Matrix](#) &)



## Related Functions

(Note that these are not member functions.)

- `const Disposable< Matrix > CholeskyDecomposition (const Matrix &m, bool flexible=false)`
- `const Disposable< Matrix > operator+ (const Matrix &, const Matrix &)`
- `const Disposable< Matrix > operator- (const Matrix &, const Matrix &)`
- `const Disposable< Matrix > operator * (const Matrix &, Real)`
- `const Disposable< Matrix > operator * (Real, const Matrix &)`
- `const Disposable< Matrix > operator/ (const Matrix &, Real)`
- `const Disposable< Array > operator * (const Array &, const Matrix &)`
- `const Disposable< Array > operator * (const Matrix &, const Array &)`
- `const Disposable< Matrix > operator * (const Matrix &, const Matrix &)`
- `const Disposable< Matrix > transpose (const Matrix &)`
- `const Disposable< Matrix > outerProduct (const Array &v1, const Array &v2)`
- `template<class Iterator1, class Iterator2> const Disposable< Matrix > outerProduct (Iterator1 v1begin, Iterator1 v1end, Iterator2 v2begin, Iterator2 v2end)`
- `void swap (Matrix &, Matrix &)`
- `std::ostream & operator<< (std::ostream &, const Matrix &)`
- `const Disposable< Matrix > pseudoSqrt (const Matrix &, SalvagingAlgorithm::Type=SalvagingAlgorithm::None)`  
*Returns the pseudo square root of a real symmetric matrix.*
- `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real componentRetainedPercentage, SalvagingAlgorithm::Type)`  
*Returns the rank-reduced pseudo square root of a real symmetric matrix.*

## 7.559.2 Member Function Documentation

### 7.559.2.1 `const Matrix & operator+= (const Matrix &)`

**Precondition:**

all matrices involved in an algebraic expression must have the same size.

## 7.559.3 Friends And Related Function Documentation

### 7.559.3.1 `const Disposable< Matrix > pseudoSqrt (const Matrix &, SalvagingAlgorithm::Type = SalvagingAlgorithm::None)` [related]

Returns the pseudo square root of a real symmetric matrix.

Given a matrix  $M$ , the result  $S$  is defined as the matrix such that  $SS^T = M$ . If the matrix is not positive semi definite, it can return 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.

**Precondition:**

the given matrix must be symmetric.

**Todo**

- implement Higham algorithm: Higham "Computing the nearest correlation matrix"

**Tests**

- the correctness of the results is tested by reproducing known good data.
- the correctness of the results is tested by checking returned values against numerical calculations.

**7.559.3.2** `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real componentRetainedPercentage, SalvagingAlgorithm::Type)` [related]

Returns the rank-reduced pseudo square root of a real symmetric matrix.

The result matrix has rank≤maxRank. If maxRank>=size, then the specified percentage of eigenvalues out of the eigenvalues' sum is retained.

If the input matrix is not positive semi definite, it can return an approximation of the pseudo square root using a (user selected) salvaging algorithm.

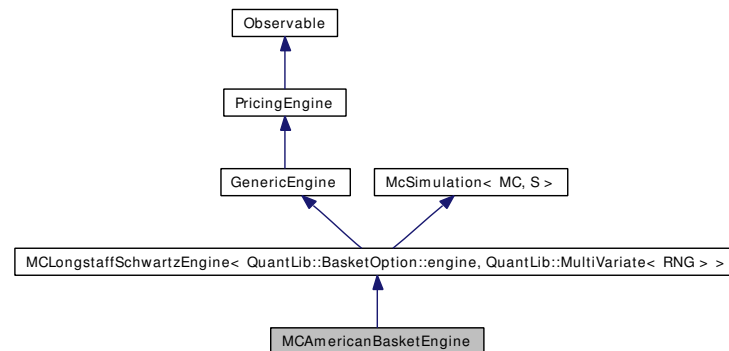
**Precondition:**

the given matrix must be symmetric.

## 7.560 MCAmericanBasketEngine Class Template Reference

```
#include <ql/PricingEngines/Basket/mcamericanbasketengine.hpp>
```

Inheritance diagram for MCAmericanBasketEngine:



### 7.560.1 Detailed Description

```
template<class RNG = PseudoRandom> class QuantLib::MCAmericanBasketEngine< RNG >
```

least-square Monte Carlo engine

#### Warning

This method is intrinsically weak for out-of-the-money options.

### Public Member Functions

- **MCAmericanBasketEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed, Size nCalibrationSamples=[Null](#)< Size >())
- **MCAmericanBasketEngine** (Size requiredSamples, Size timeSteps, BigNatural seed=0, bool antitheticSampling=false)

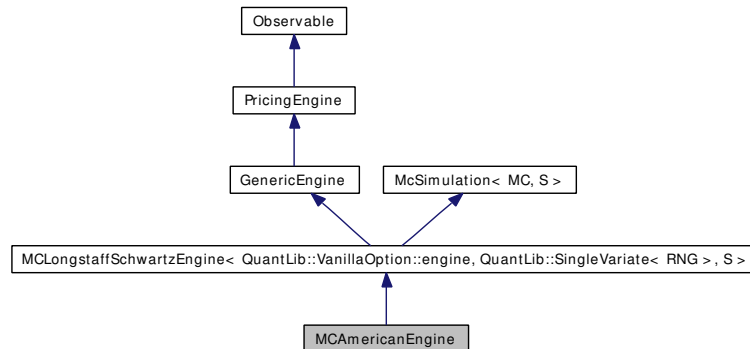
### Protected Member Functions

- boost::shared\_ptr< [LongstaffSchwartzPathPricer](#)< [MultiPath](#) > > **lsmPathPricer** () const

## 7.561 MCAmericanEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mcamericanengine.hpp>
```

Inheritance diagram for MCAmericanEngine:



### 7.561.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCAmericanEngine< RNG, S >
```

American Monte Carlo engine.

References:

#### Tests

the correctness of the returned value is tested by reproducing results available in web/literature

### Public Member Functions

- **MCAmericanEngine** (Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, Big-Natural seed, Size polynomOrder, LsmBasisSystem::PolynomType polynomType, Size n-CalibrationSamples=[Null](#)< Size >())

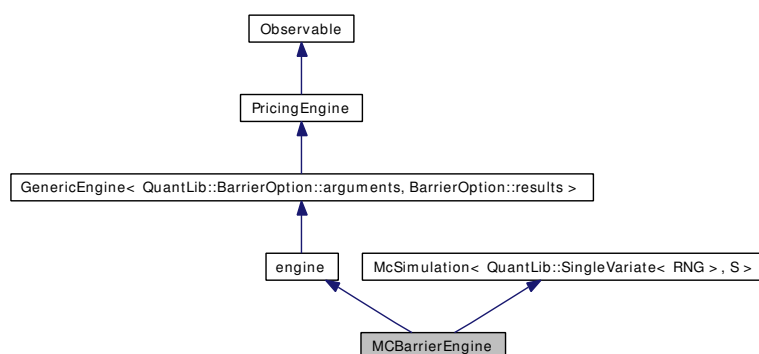
### Protected Member Functions

- boost::shared\_ptr< [LongstaffSchwartzPathPricer](#)< [Path](#) > > **lsmPathPricer** () const
- Real **controlVariateValue** () const
- boost::shared\_ptr< [PricingEngine](#) > **controlPricingEngine** () const
- boost::shared\_ptr< [PathPricer](#)< [Path](#) > > **controlPathPricer** () const

## 7.562 MBarrierEngine Class Template Reference

```
#include <ql/PricingEngines/Barrier/mcbarrierengine.hpp>
```

Inheritance diagram for MBarrierEngine:



### 7.562.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MBarrierEngine< RNG, S >
```

Pricing engine for barrier options using Monte Carlo simulation.

Uses the Brownian-bridge correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic Option Valuation* - D.R. Beaglehole, P.H. Dybvig and G. Zhou *Financial Analysts Journal*; Jan/Feb 1997; 53, 1. pg. 62-68 and *Simulating path-dependent options: A new approach* - M. El Babsiri and G. Noel *Journal of Derivatives*; Winter 1998; 6, 2; pg. 65-83

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

### Public Types

- typedef `McSimulation< SingleVariate< RNG >, S >::path_generator_type` **path\_generator\_type**
- typedef `McSimulation< SingleVariate< RNG >, S >::path_pricer_type` **path\_pricer\_type**
- typedef `McSimulation< SingleVariate< RNG >, S >::stats_type` **stats\_type**

### Public Member Functions

- **MBarrierEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, bool isBiased, BigNatural seed)
- void **calculate** () const

## Protected Member Functions

- [TimeGrid](#) **timeGrid** () const
- boost::shared\_ptr< path\_generator\_type > **pathGenerator** () const
- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const

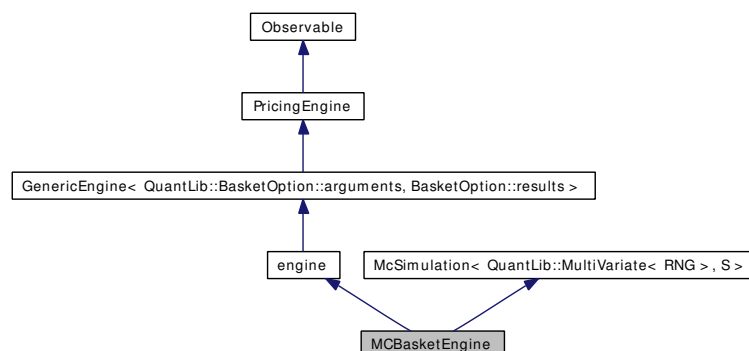
## Protected Attributes

- Size **maxTimeStepsPerYear\_**
- Size **requiredSamples\_**
- Size **maxSamples\_**
- Real **requiredTolerance\_**
- bool **isBiased\_**
- bool **brownianBridge\_**
- BigNatural **seed\_**

## 7.563 MCBasketEngine Class Template Reference

```
#include <ql/PricingEngines/Basket/mcbasketengine.hpp>
```

Inheritance diagram for MCBasketEngine:



### 7.563.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCBasketEngine< RNG, S >
```

Pricing engine for basket options using Monte Carlo simulation.

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

### Public Types

- typedef [McSimulation](#)< [MultiVariate](#)< RNG >, S >::path\_generator\_type **path\_generator\_type**
- typedef [McSimulation](#)< [MultiVariate](#)< RNG >, S >::path\_pricer\_type **path\_pricer\_type**
- typedef [McSimulation](#)< [MultiVariate](#)< RNG >, S >::stats\_type **stats\_type**

### Public Member Functions

- **MCBasketEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const

### Protected Member Functions

- [TimeGrid](#) **timeGrid** () const
- boost::shared\_ptr< path\_generator\_type > **pathGenerator** () const
- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const

## Protected Attributes

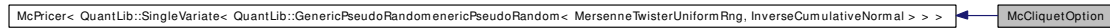
- Size `maxTimeStepsPerYear_`
- Size `requiredSamples_`
- Size `maxSamples_`
- Real `requiredTolerance_`
- bool `brownianBridge_`
- BigNatural `seed_`



## 7.564 McCliquetOption Class Reference

```
#include <ql/Pricers/mccliquetoption.hpp>
```

Inheritance diagram for McCliquetOption:



### 7.564.1 Detailed Description

simple example of Monte Carlo pricer

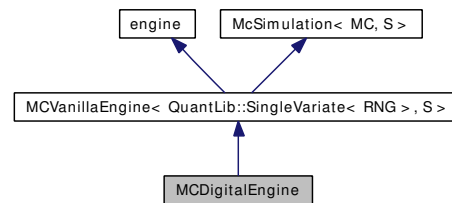
#### Public Member Functions

- **McCliquetOption** (Option::Type type, [Real](#) underlying, [Real](#) moneyiness, const [Handle](#)< [YieldTermStructure](#) > &dividendYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > &times, [Real](#) accruedCoupon, [Real](#) lastFixing, [Real](#) localCap, [Real](#) localFloor, [Real](#) globalCap, [Real](#) globalFloor, bool redemptionOnly, BigNatural seed=0)

## 7.565 MCDigitalEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mcdigitalengine.hpp>
```

Inheritance diagram for MCDigitalEngine:



### 7.565.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDigital-
Engine< RNG, S >
```

Pricing engine for digital options using Monte Carlo simulation.

Uses the Brownian [Bridge](#) correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic [Option](#) Valuation* - D.R. Beaglehole, P.H. Dybvig and G. Zhou *Financial Analysts Journal*; Jan/Feb 1997; 53, 1. pg. 62-68 and *Simulating path-dependent options: A new approach* - M. El Babsiri and G. Noel *Journal of Derivatives*; Winter 1998; 6, 2; pg. 65-83

#### Tests

the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing known good results.

### Public Types

- typedef [MCVanillaEngine](#)< [SingleVariate](#)< RNG > , S >::path\_generator\_type **path\_generator\_type**
- typedef [MCVanillaEngine](#)< [SingleVariate](#)< RNG > , S >::path\_pricer\_type **path\_pricer\_type**
- typedef [MCVanillaEngine](#)< [SingleVariate](#)< RNG > , S >::stats\_type **stats\_type**

### Public Member Functions

- **MCDigitalEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

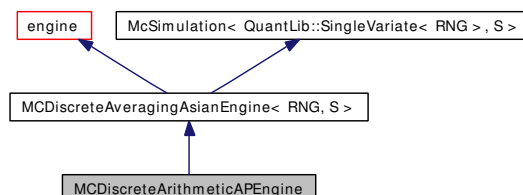
### Protected Member Functions

- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const

## 7.566 MCDiscreteArithmeticAPEngine Class Template Reference

```
#include <ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp>
```

Inheritance diagram for MCDiscreteArithmeticAPEngine:



### 7.566.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscreteArithmeticAPEngine< RNG, S >
```

Monte Carlo pricing engine for discrete arithmetic average price Asian.

Monte Carlo pricing engine for discrete arithmetic average price Asian options. It can use [MCDiscreteGeometricAPEngine](#) (Monte Carlo discrete arithmetic average price engine) and [AnalyticDiscreteGeometricAveragePriceAsianEngine](#) (analytic discrete arithmetic average price engine) for control variation.

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

### Public Types

- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::path\_generator\_type **path\_generator\_type**
- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::path\_pricer\_type **path\_pricer\_type**
- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::stats\_type **stats\_type**

### Public Member Functions

- **MCDiscreteArithmeticAPEngine** (Size maxTimeStepPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural)

### Protected Member Functions

- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const
- boost::shared\_ptr< path\_pricer\_type > **controlPathPricer** () const
- boost::shared\_ptr< [PricingEngine](#) > **controlPricingEngine** () const

## 7.567 McDiscreteArithmeticASO Class Reference

```
#include <ql/Pricers/mcdiscretearithmeticaso.hpp>
```

Inheritance diagram for McDiscreteArithmeticASO:



### 7.567.1 Detailed Description

example of Monte Carlo pricer using a control variate.

#### Todo

continous-averaging version

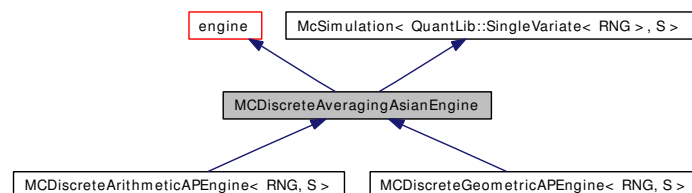
### Public Member Functions

- **McDiscreteArithmeticASO** (Option::Type type, [Real](#) underlying, const [Handle](#)< [YieldTermStructure](#) > &dividendYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > &times, bool controlVariate, BigNatural seed=0)

## 7.568 MCDiscreteAveragingAsianEngine Class Template Reference

```
#include <ql/PricingEngines/Asian/mcdiscreteasianengine.hpp>
```

Inheritance diagram for MCDiscreteAveragingAsianEngine:



### 7.568.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscrete-
AveragingAsianEngine< RNG, S >
```

Pricing engine for discrete average Asians using Monte Carlo simulation.

#### Warning

control-variate calculation is disabled under VC++6.

### Public Types

- typedef [McSimulation](#)< [SingleVariate](#)< RNG >, S >::path\_generator\_type **path\_generator\_type**
- typedef [McSimulation](#)< [SingleVariate](#)< RNG >, S >::path\_pricer\_type **path\_pricer\_type**
- typedef [McSimulation](#)< [SingleVariate](#)< RNG >, S >::stats\_type **stats\_type**

### Public Member Functions

- **MCDiscreteAveragingAsianEngine** (Size maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const

### Protected Member Functions

- [TimeGrid](#) **timeGrid** () const
- boost::shared\_ptr< path\_generator\_type > **pathGenerator** () const
- Real **controlVariateValue** () const

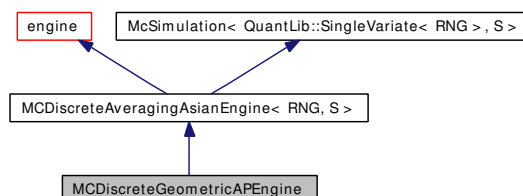
## Protected Attributes

- Size `maxTimeStepsPerYear_`
- Size `requiredSamples_`
- Size `maxSamples_`
- Real `requiredTolerance_`
- bool `brownianBridge_`
- BigNatural `seed_`

## 7.569 MCDiscreteGeometricAPEngine Class Template Reference

```
#include <ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp>
```

Inheritance diagram for MCDiscreteGeometricAPEngine:



### 7.569.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscreteGeometricAPEngine< RNG, S >
```

Monte Carlo pricing engine for discrete geometric average price Asian.

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

### Public Types

- typedef [MCDiscreteAveragingAsianEngine< RNG, S >::path\\_generator\\_type](#) **path\_generator\_type**
- typedef [MCDiscreteAveragingAsianEngine< RNG, S >::path\\_pricer\\_type](#) **path\_pricer\_type**
- typedef [MCDiscreteAveragingAsianEngine< RNG, S >::stats\\_type](#) **stats\_type**

### Public Member Functions

- **MCDiscreteGeometricAPEngine** (Size maxTimeStepPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

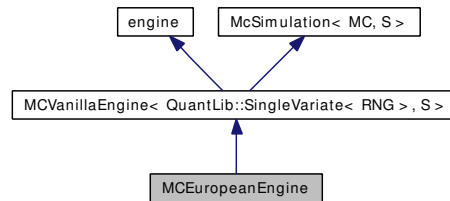
### Protected Member Functions

- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const

## 7.570 MCEuropeanEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mceuropeanengine.hpp>
```

Inheritance diagram for MCEuropeanEngine:



### 7.570.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCEuropean-
Engine< RNG, S >
```

European option pricing engine using Monte Carlo simulation.

#### Tests

the correctness of the returned value is tested by checking it against analytic results.

### Public Types

- typedef [MCVanillaEngine](#)< [SingleVariate](#)< RNG >, S >::path\_generator\_type **path\_generator\_type**
- typedef [MCVanillaEngine](#)< [SingleVariate](#)< RNG >, S >::path\_pricer\_type **path\_pricer\_type**
- typedef [MCVanillaEngine](#)< [SingleVariate](#)< RNG >, S >::stats\_type **stats\_type**

### Public Member Functions

- **MCEuropeanEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

### Protected Member Functions

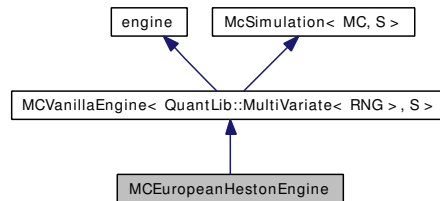
- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const



## 7.571 MCEuropeanHestonEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp>
```

Inheritance diagram for MCEuropeanHestonEngine:



### 7.571.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCEuropeanHestonEngine< RNG, S >
```

Monte Carlo Heston-model engine for European options.

#### Tests

the correctness of the returned value is tested by reproducing results available in web/literature

### Public Types

- typedef [MCVanillaEngine](#)< [MultiVariate](#)< RNG >, S >::path\_pricer\_type **path\_pricer\_type**

### Public Member Functions

- **MCEuropeanHestonEngine** (Size timeSteps, Size timeStepsPerYear, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)

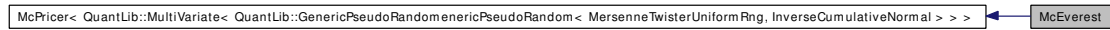
### Protected Member Functions

- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const

## 7.572 McEverest Class Reference

```
#include <ql/Pricers/mceverest.hpp>
```

Inheritance diagram for McEverest:



### 7.572.1 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 std::vector< [Handle< YieldTermStructure >](#) &dividendYield, const [Handle< YieldTermStructure >](#) &riskFreeRate, const std::vector< [Handle< BlackVolTermStructure >](#) &volatilities, const [Matrix](#) &correlation, [Time](#) residualTime, [BigNatural](#) seed=0)

## 7.573 McHimalaya Class Reference

```
#include <ql/Pricers/mchimalaya.hpp>
```

Inheritance diagram for McHimalaya:



### 7.573.1 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.

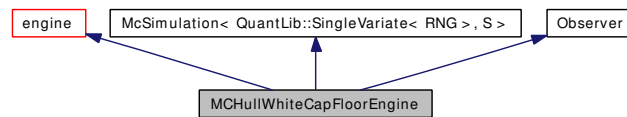
### Public Member Functions

- **McHimalaya** (const std::vector< [Real](#) > &underlyings, const std::vector< [Handle](#)< [YieldTermStructure](#) > > &dividendYields, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const std::vector< [Handle](#)< [BlackVolTermStructure](#) > > &volatilities, const [Matrix](#) &correlation, [Real](#) strike, const std::vector< [Time](#) > &times, BigNatural seed=0)

## 7.574 MCHullWhiteCapFloorEngine Class Template Reference

```
#include <ql/PricingEngines/CapFloor/mchullwhiteengine.hpp>
```

Inheritance diagram for MCHullWhiteCapFloorEngine:



### 7.574.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCHullWhiteCapFloorEngine< RNG, S >
```

Monte Carlo Hull-White engine for cap/floors.

### Public Types

- typedef simulation::path\_generator\_type **path\_generator\_type**
- typedef simulation::path\_pricer\_type **path\_pricer\_type**
- typedef simulation::stats\_type **stats\_type**

### Public Member Functions

- **MCHullWhiteCapFloorEngine** (const boost::shared\_ptr< [HullWhite](#) > &model, bool brownianBridge, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const
- void [update](#) ()

### Protected Member Functions

- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const
- [TimeGrid](#) **timeGrid** () const
- boost::shared\_ptr< path\_generator\_type > **pathGenerator** () const

## 7.574.2 Member Function Documentation

### 7.574.2.1 void update () [virtual]

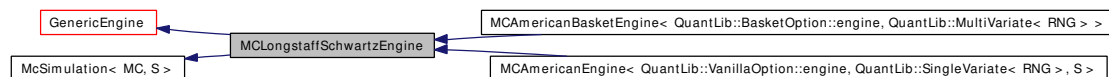
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.575 MCLongstaffSchwartzEngine Class Template Reference

```
#include <ql/PricingEngines/mclongstaffschwartzengine.hpp>
```

Inheritance diagram for MCLongstaffSchwartzEngine:



### 7.575.1 Detailed Description

```
template<class GenericEngine, class MC, class S = Statistics> class QuantLib::MCLongstaffSchwartzEngine< GenericEngine, MC, S >
```

Longstaff Schwarz Monte Carlo engine for early exercise options.

References:

Francis Longstaff, Eduardo Schwartz, 2001. Valuing American Options by Simulation: A Simple Least-Squares Approach, The Review of Financial Studies, Volume 14, No. 1, 113-147

#### Tests

the correctness of the returned value is tested by reproducing results available in web/literature

### Public Types

- typedef MC::path\_type **path\_type**
- typedef [McSimulation](#)< MC, S >::stats\_type **stats\_type**
- typedef [McSimulation](#)< MC, S >::path\_pricer\_type **path\_pricer\_type**
- typedef [McSimulation](#)< MC, S >::path\_generator\_type **path\_generator\_type**

### Public Member Functions

- **MCLongstaffSchwartzEngine** (Size timeSteps, Size timeStepsPerYear, bool brownian-Bridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real required-Tolerance, Size maxSamples, BigNatural seed, Size nCalibrationSamples=[Null](#)< Size >())
- void **calculate** () const

### Protected Member Functions

- virtual boost::shared\_ptr< [LongstaffSchwartzPathPricer](#)< path\_type > > **lsmPathPricer** () const=0
- [TimeGrid](#) **timeGrid** () const
- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const
- boost::shared\_ptr< path\_generator\_type > **pathGenerator** () const

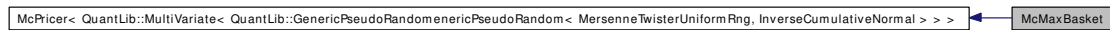
## Protected Attributes

- const Size **timeSteps\_**
- const Size **timeStepsPerYear\_**
- const bool **brownianBridge\_**
- const Size **requiredSamples\_**
- const Real **requiredTolerance\_**
- const Size **maxSamples\_**
- const Size **seed\_**
- const Size **nCalibrationSamples\_**
- boost::shared\_ptr< [LongstaffSchwartzPathPricer](#)< path\_type > > **pathPricer\_**

## 7.576 McMaxBasket Class Reference

```
#include <ql/Pricers/mcmaxbasket.hpp>
```

Inheritance diagram for McMaxBasket:



### 7.576.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

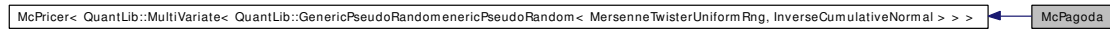
#### Public Member Functions

- **McMaxBasket** (const std::vector< [Real](#) > &underlyings, const std::vector< [Handle](#)< [YieldTermStructure](#) > > &dividendYields, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const std::vector< [Handle](#)< [BlackVolTermStructure](#) > > &volatilities, const [Matrix](#) &correlation, [Time](#) residualTime, [BigNatural](#) seed=0)

## 7.577 McPagoda Class Reference

```
#include <ql/Pricers/mcpagoda.hpp>
```

Inheritance diagram for McPagoda:



### 7.577.1 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

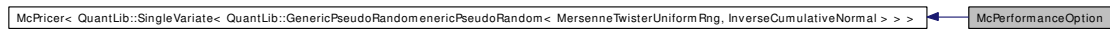
- **McPagoda** (const std::vector< Real > &underlyings, Real fraction, Real roof, const std::vector< Handle< YieldTermStructure > > &dividendYields, const Handle< YieldTermStructure > &riskFreeRate, const std::vector< Handle< BlackVolTermStructure > > &volatilities, const Matrix &correlation, const std::vector< Time > &times, BigNatural seed=0)



## 7.578 McPerformanceOption Class Reference

```
#include <ql/Pricers/mcperformanceoption.hpp>
```

Inheritance diagram for McPerformanceOption:



### 7.578.1 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 \$  $\max(S/X - 1)$  \$.

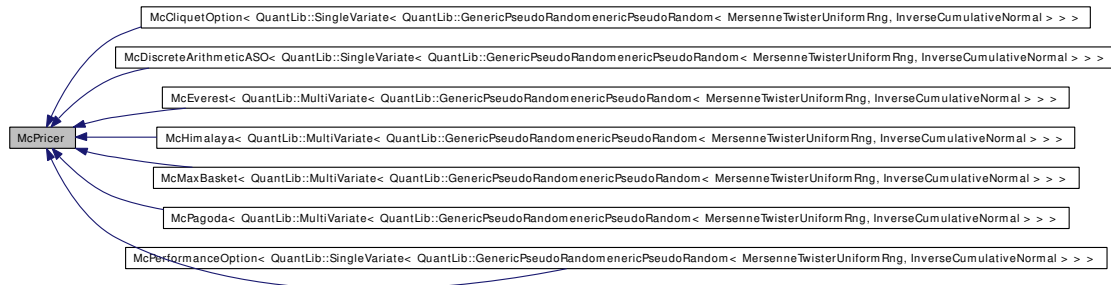
### Public Member Functions

- **McPerformanceOption** (Option::Type type, [Real](#) underlying, [Real](#) moneyiness, const [Handle](#)< [YieldTermStructure](#) > &dividendYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > &times, BigNatural seed=0)

## 7.579 McPricer Class Template Reference

```
#include <ql/Pricers/mcpricer.hpp>
```

Inheritance diagram for McPricer:



### 7.579.1 Detailed Description

```
template<class MC, class S = Statistics> class QuantLib::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](#) gives an easy way to write a Monte Carlo Pricer. See [McEuropean](#) as example of one factor pricer, [Basket](#) as example of multi factor pricer.

### Public Member Functions

- Real [value](#) (Real tolerance, Size maxSamples=QL\_MAX\_INTEGER, Size minSamples=1023) const  
*add samples until the required tolerance is reached*
- Real [valueWithSamples](#) (Size samples, Size minSamples=1023) const  
*simulate a fixed number of samples*
- Real [errorEstimate](#) () const  
*estimated error of the samples simulated so far*
- const S & [sampleAccumulator](#) (void) const  
*access to the sample accumulator for more statistics*

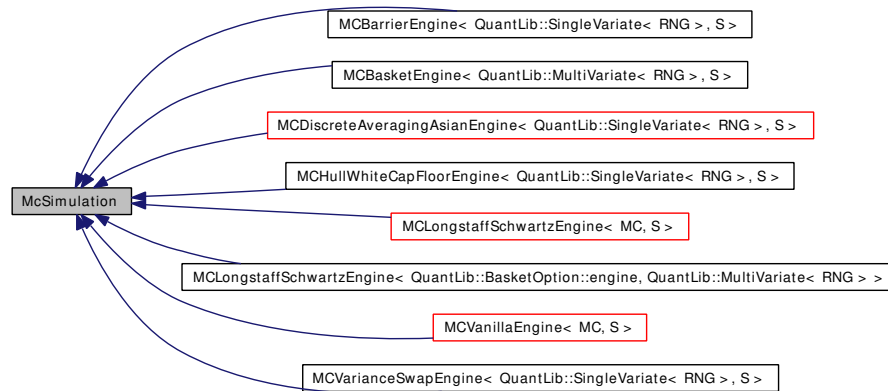
### Protected Attributes

- boost::shared\_ptr< [MonteCarloModel](#)< MC, S > > [mcModel\\_](#)

## 7.580 McSimulation Class Template Reference

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

Inheritance diagram for McSimulation:



### 7.580.1 Detailed Description

```
template<class MC, class S = Statistics> class QuantLib::McSimulation< MC, S >
```

base class for Monte Carlo engines

Eventually this class might offer greeks methods. Deriving a class from [McSimulation](#) gives an easy way to write a Monte Carlo engine.

See [McVanillaEngine](#) as an example.

### Public Types

- typedef [MonteCarloModel](#)< MC, S >::path\_generator\_type **path\_generator\_type**
- typedef [MonteCarloModel](#)< MC, S >::path\_pricer\_type **path\_pricer\_type**
- typedef [MonteCarloModel](#)< MC, S >::stats\_type **stats\_type**

### Public Member Functions

- Real [value](#) (Real tolerance, Size maxSamples=QL\_MAX\_INTEGER, Size minSamples=1023) const  
*add samples until the required absolute tolerance is reached*
- Real [valueWithSamples](#) (Size samples) const  
*simulate a fixed number of samples*
- Real [errorEstimate](#) () const  
*error estimated using the samples simulated so far*
- const stats\_type & [sampleAccumulator](#) (void) const

*access to the sample accumulator for richer statistics*

- void [calculate](#) (Real requiredTolerance, Size requiredSamples, Size maxSamples) const  
*basic calculate method provided to inherited pricing engines*

## Protected Member Functions

- **McSimulation** (bool antitheticVariate, bool controlVariate)
- virtual boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const=0
- virtual boost::shared\_ptr< path\_generator\_type > **pathGenerator** () const=0
- virtual [TimeGrid](#) **timeGrid** () const=0
- virtual boost::shared\_ptr< path\_pricer\_type > **controlPathPricer** () const
- virtual boost::shared\_ptr< [PricingEngine](#) > **controlPricingEngine** () const
- virtual Real **controlVariateValue** () const

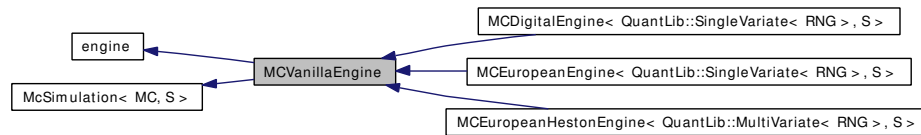
## Protected Attributes

- boost::shared\_ptr< [MonteCarloModel](#)< MC, S > > **mcModel\_**
- bool **antitheticVariate\_**
- bool **controlVariate\_**

## 7.581 MCVanillaEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

Inheritance diagram for MCVanillaEngine:



### 7.581.1 Detailed Description

```
template<class MC, class S = Statistics> class QuantLib::MCVanillaEngine< MC, S >
```

Pricing engine for vanilla options using Monte Carlo simulation.

#### Public Member Functions

- void **calculate** () const

#### Protected Types

- typedef [McSimulation](#)< MC, S >::path\_generator\_type **path\_generator\_type**
- typedef [McSimulation](#)< MC, S >::path\_pricer\_type **path\_pricer\_type**
- typedef [McSimulation](#)< MC, S >::stats\_type **stats\_type**

#### Protected Member Functions

- **MCVanillaEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- [TimeGrid](#) **timeGrid** () const
- boost::shared\_ptr< path\_generator\_type > **pathGenerator** () const
- Real **controlVariateValue** () const

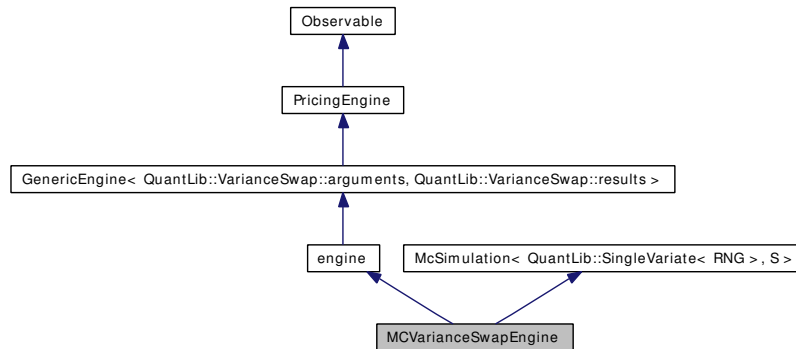
#### Protected Attributes

- Size **timeSteps\_**
- Size **timeStepsPerYear\_**
- Size **requiredSamples\_**
- Size **maxSamples\_**
- Real **requiredTolerance\_**
- bool **brownianBridge\_**
- BigNatural **seed\_**

## 7.582 MCVarianceSwapEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/mcvarianceswapengine.hpp>
```

Inheritance diagram for MCVarianceSwapEngine:



### 7.582.1 Detailed Description

**template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCVarianceSwapEngine< RNG, S >**

Variance-swap pricing engine using Monte Carlo simulation,.

as described in Demeterfi, Derman, Kamal & Zou, "A Guide to Volatility and Variance Swaps", 1999

#### Todo

define tolerance of numerical integral and incorporate it in errorEstimate

#### Tests

returned fair variances checked for consistency with implied volatility curve.

### Public Types

- typedef [McSimulation](#)< [SingleVariate](#)< RNG >, S >::path\_generator\_type **path\_generator\_type**
- typedef [McSimulation](#)< [SingleVariate](#)< RNG >, S >::path\_pricer\_type **path\_pricer\_type**
- typedef [McSimulation](#)< [SingleVariate](#)< RNG >, S >::stats\_type **stats\_type**

### Public Member Functions

- **MCVarianceSwapEngine** (Size timeSteps, Size timeStepsPerYear, bool brownianBridge, bool antitheticVariate, Size requiredSamples, Real requiredTolerance, Size maxSamples, BigNatural seed)
- void **calculate** () const

## Protected Member Functions

- `boost::shared_ptr< path_pricer_type > pathPricer () const`
- `TimeGrid timeGrid () const`
- `boost::shared_ptr< path_generator_type > pathGenerator () const`

## Protected Attributes

- `Size timeSteps_`
- `Size timeStepsPerYear_`
- `Size requiredSamples_`
- `Size maxSamples_`
- `Real requiredTolerance_`
- `bool brownianBridge_`
- `BigNatural seed_`

## 7.583 MersenneTwisterUniformRng Class Reference

```
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

### 7.583.1 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/matsumoto/emt.html>

#### Tests

the correctness of the returned values is tested by checking them against known good results.

### Public Types

- typedef [Sample](#)< [Real](#) > **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*

### 7.583.2 Constructor & Destructor Documentation

7.583.2.1 [MersenneTwisterUniformRng](#) (unsigned long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

### 7.583.3 Member Function Documentation

7.583.3.1 [sample\\_type](#) next () const

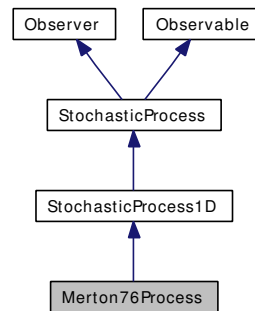
returns a sample with weight 1.0 containing a random number on (0.0, 1.0)-real-interval



## 7.584 Merton76Process Class Reference

```
#include <ql/Processes/merton76process.hpp>
```

Inheritance diagram for Merton76Process:



### 7.584.1 Detailed Description

Merton-76 jump-diffusion process.

#### Public Member Functions

- **Merton76Process** (const [Handle](#)< [Quote](#) > &stateVariable, const [Handle](#)< [YieldTermStructure](#) > &dividendTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &blackVolTS, const [Handle](#)< [Quote](#) > &jumpInt, const [Handle](#)< [Quote](#) > &logJMean, const [Handle](#)< [Quote](#) > &logJVol, const boost::shared\_ptr< discretization > &d=boost::shared\_ptr< discretization >(new [EulerDiscretization](#)))
- Time [time](#) (const [Date](#) &) const

#### StochasticProcess1D interface

- Real [x0](#) () const  
*returns the initial value of the state variable*
- Real [drift](#) (Time, Real) const  
*returns the drift part of the equation, i.e.  $\mu(t, x_t)$*
- Real [diffusion](#) (Time, Real) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*
- Real [apply](#) (Real, Real) const

#### Inspectors

- const boost::shared\_ptr< [Quote](#) > & [stateVariable](#) () const
- const boost::shared\_ptr< [YieldTermStructure](#) > & [dividendYield](#) () const
- const boost::shared\_ptr< [YieldTermStructure](#) > & [riskFreeRate](#) () const
- const boost::shared\_ptr< [BlackVolTermStructure](#) > & [blackVolatility](#) () const
- const boost::shared\_ptr< [Quote](#) > & [jumpIntensity](#) () const
- const boost::shared\_ptr< [Quote](#) > & [logMeanJump](#) () const
- const boost::shared\_ptr< [Quote](#) > & [logJumpVolatility](#) () const

## 7.584.2 Member Function Documentation

### 7.584.2.1 `Real apply (Real, Real) const` [virtual]

applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented from [StochasticProcess1D](#).

### 7.584.2.2 `Time time (const Date &) const` [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

**Note:**

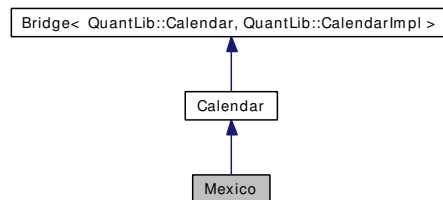
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

## 7.585 Mexico Class Reference

```
#include <ql/Calendars/mexico.hpp>
```

Inheritance diagram for Mexico:



### 7.585.1 Detailed Description

Mexican calendars

Holidays for the Mexican stock exchange (data from <http://www.bmv.com.mx/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Constitution Day, February 5th
- Birthday of Benito Juarez, March 21st
- Holy Thursday
- Good Friday
- Labour Day, May 1st
- National Day, September 16th
- Our Lady of Guadalupe, December 12th
- Christmas, December 25th

### Public Types

- enum [Market](#) { [BMV](#) }

### Public Member Functions

- [Mexico](#) ([Market](#) m=BMV)

## 7.585.2 Member Enumeration Documentation

### 7.585.2.1 enum [Market](#)

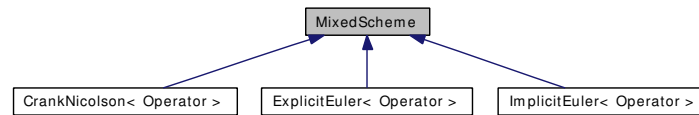
Enumerator:

*BMV* Mexican stock exchange.

## 7.586 MixedScheme Class Template Reference

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Inheritance diagram for MixedScheme:



### 7.586.1 Detailed Description

```
template<class Operator> class QuantLib::MixedScheme< Operator >
```

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

In this implementation, the passed operator must be derived from either `TimeConstantOperator` or `TimeDependentOperator`. Also, it must implement at least the following interface:

```

typedef ... array_type;

// 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
array_type applyTo(const array_type&);
array_type solveFor(const array_type&);
static Operator identity(Size size);

// operator algebra
Operator operator*(Real, 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

- derive variable theta schemes
- introduce multi time-level schemes.

### Public Types

- `typedef OperatorTraits< Operator > traits`
- `typedef traits::operator_type operator_type`

- typedef traits::array\_type **array\_type**
- typedef traits::bc\_set **bc\_set**
- typedef [traits::condition\\_type](#) **condition\_type**

## Public Member Functions

- **MixedScheme** (const operator\_type &L, Real theta, const bc\_set &bcs)
- void **step** (array\_type &a, Time t)
- void **setStep** (Time dt)

## Protected Attributes

- operator\_type **L\_**
- operator\_type **I\_**
- operator\_type **explicitPart\_**
- operator\_type **implicitPart\_**
- Time **dt\_**
- Real **theta\_**
- bc\_set **bcs\_**

## 7.587 Money Class Reference

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

### 7.587.1 Detailed Description

amount of cash

#### Tests

money arithmetic is tested with and without currency conversions.

### Conversion settings

These parameters are used for combining money amounts in different currencies

- enum [ConversionType](#) { [NoConversion](#), [BaseCurrencyConversion](#), [AutomatedConversion](#) }
- static [ConversionType](#) [conversionType](#)
- static [Currency](#) [baseCurrency](#)

### Public Member Functions

#### Inspectors

- const [Currency](#) & [currency](#) () const
- Decimal [value](#) () const
- [Money](#) [rounded](#) () const

#### Money arithmetics

See below for non-member functions and for settings which determine the behavior of the operators.

- [Money](#) [operator+](#) () const
- [Money](#) [operator-](#) () const
- [Money](#) & [operator+=](#) (const [Money](#) &)
- [Money](#) & [operator-=](#) (const [Money](#) &)
- [Money](#) & [operator\\*=](#) (Decimal)
- [Money](#) & [operator/=](#) (Decimal)

### Related Functions

(Note that these are not member functions.)

- [Money](#) [operator+](#) (const [Money](#) &, const [Money](#) &)
- [Money](#) [operator-](#) (const [Money](#) &, const [Money](#) &)
- [Money](#) [operator\\*](#) (const [Money](#) &, Decimal)
- [Money](#) [operator\\*](#) (Decimal, const [Money](#) &)
- [Money](#) [operator/](#) (const [Money](#) &, Decimal)
- Decimal [operator/](#) (const [Money](#) &, const [Money](#) &)
- bool [operator==](#) (const [Money](#) &, const [Money](#) &)

- `bool operator!=` (const [Money](#) &, const [Money](#) &)
- `bool operator<` (const [Money](#) &, const [Money](#) &)
- `bool operator<=` (const [Money](#) &, const [Money](#) &)
- `bool operator>` (const [Money](#) &, const [Money](#) &)
- `bool operator>=` (const [Money](#) &, const [Money](#) &)
- `bool close` (const [Money](#) &, const [Money](#) &, Size n=42)
- `bool close_enough` (const [Money](#) &, const [Money](#) &, Size n=42)
- `Money operator *` (Decimal, const [Currency](#) &)
- `Money operator *` (const [Currency](#) &, Decimal)
- `std::ostream & operator<<` (std::ostream &, const [Money](#) &)

## 7.587.2 Member Enumeration Documentation

### 7.587.2.1 enum [ConversionType](#)

Enumerator:

*NoConversion* do not perform conversions

*BaseCurrencyConversion* convert both operands to the base currency before converting

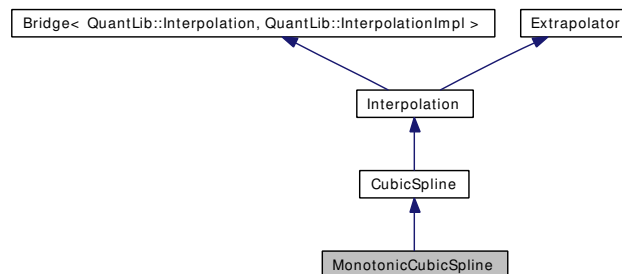
*AutomatedConversion* return the result in the currency of the first operand



## 7.588 MonotonicCubicSpline Class Reference

```
#include <ql/Math/cubicspline.hpp>
```

Inheritance diagram for MonotonicCubicSpline:



### 7.588.1 Detailed Description

Cubic spline with monotonicity constraint

#### Public Member Functions

- `template<class I1, class I2> MonotonicCubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, CubicSpline::BoundaryCondition leftCondition, Real leftConditionValue, CubicSpline::BoundaryCondition rightCondition, Real rightConditionValue)`

### 7.588.2 Constructor & Destructor Documentation

- 7.588.2.1 `MonotonicCubicSpline (const I1 & xBegin, const I1 & xEnd, const I2 & yBegin, CubicSpline::BoundaryCondition leftCondition, Real leftConditionValue, CubicSpline::BoundaryCondition rightCondition, Real rightConditionValue)`

#### Precondition:

the  $x$  values must be sorted.

## 7.589 MonteCarloModel Class Template Reference

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

### 7.589.1 Detailed Description

```
template<class mc_traits, class stats_traits = Statistics> class QuantLib::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 [mctrails.hpp](#) 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**

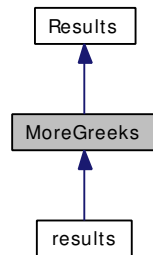
### Public Member Functions

- **MonteCarloModel** (const boost::shared\_ptr< path\_generator\_type > &pathGenerator, const boost::shared\_ptr< path\_pricer\_type > &pathPricer, const stats\_type &sampleAccumulator, bool antitheticVariate, const boost::shared\_ptr< path\_pricer\_type > &cvPathPricer=boost::shared\_ptr< path\_pricer\_type >(), result\_type cvOptionValue=result\_type())
- void **addSamples** (Size samples)
- const stats\_type & **sampleAccumulator** (void) const

## 7.590 MoreGreeks Class Reference

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

Inheritance diagram for MoreGreeks:



### 7.590.1 Detailed Description

more additional option results

#### Public Member Functions

- void **reset** ()

#### Public Attributes

- Real **itmCashProbability**
- Real **deltaForward**
- Real **elasticity**
- Real **thetaPerDay**
- Real **strikeSensitivity**

## 7.591 MoroInverseCumulativeNormal Class Reference

```
#include <ql/Math/normaldistribution.hpp>
```

### 7.591.1 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::InverseCumulativeNormal](#)

### Public Member Functions

- **MoroInverseCumulativeNormal** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real  $x$ ) const

## 7.592 MTBrownianGenerator Class Reference

```
#include <ql/MarketModels/BrownianGenerators/mtbrowniangenerator.hpp>
```

### 7.592.1 Detailed Description

Incremental Brownian generator using a Mersenne-twister uniform generator and inverse-cumulative Gaussian method.

**Note:**

At this time, generation of the underlying uniform sequence is eager, while its transformation into Gaussian variates is lazy. Further optimization might be possible by using the Mersenne twister directly instead of a [RandomSequenceGenerator](#); however, it is not clear how much of a difference this would make when compared to the inverse-cumulative Gaussian calculation.

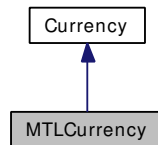
### Public Member Functions

- **MTBrownianGenerator** (Size factors, Size steps, unsigned long seed=0)
- **Real** nextStep (**Array** &)
- **Real** nextPath ()
- Size **numberOfFactors** () const
- Size **numberOfSteps** () const

## 7.593 MTLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for MTLCurrency:



### 7.593.1 Detailed Description

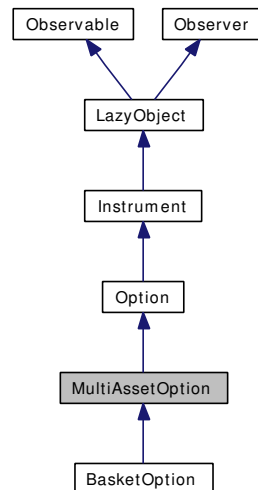
Maltese lira.

The ISO three-letter code is MTL; the numeric code is 470. It is divided in 100 cents.

## 7.594 MultiAssetOption Class Reference

```
#include <ql/Instruments/multiassetoption.hpp>
```

Inheritance diagram for MultiAssetOption:



### 7.594.1 Detailed Description

Base class for options on multiple assets.

#### Public Member Functions

- **MultiAssetOption** (const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [Payoff](#) > &, const boost::shared\_ptr< [Exercise](#) > &, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) \*) const
- void [fetchResults](#) (const [Results](#) \*) const

#### Instrument interface

- bool [isExpired](#) () const  
*returns whether the instrument is still tradable.*

#### greeks

- Real [delta](#) () const
- Real [gamma](#) () const
- Real [theta](#) () const
- Real [vega](#) () const
- Real [rho](#) () const
- Real [dividendRho](#) () const

## Protected Member Functions

- void [setupExpired](#) () const

## Protected Attributes

- Real [delta](#)\_
- Real [gamma](#)\_
- Real [theta](#)\_
- Real [vega](#)\_
- Real [rho](#)\_
- Real [dividendRho](#)\_
- boost::shared\_ptr< [StochasticProcess](#) > [stochasticProcess](#)\_

## Classes

- class [arguments](#)  
*Arguments for multi-asset option calculation*
- class [results](#)  
*Results from multi-asset option calculation*

## 7.594.2 Member Function Documentation

### 7.594.2.1 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](#).

Reimplemented in [BasketOption](#).

### 7.594.2.2 void [fetchResults](#) (const [Results](#) \*) const [virtual]

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

Reimplemented from [Instrument](#).

### 7.594.2.3 void [setupExpired](#) () const [protected, virtual]

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

Reimplemented from [Instrument](#).



## 7.595 MultiAssetOption::arguments Class Reference

```
#include <ql/Instruments/multiassetoption.hpp>
```

### 7.595.1 Detailed Description

Arguments for multi-asset option calculation

#### Public Member Functions

- void **validate** () const

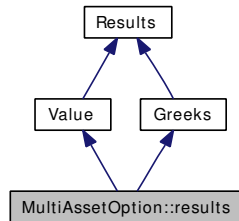
#### Public Attributes

- boost::shared\_ptr< [StochasticProcess](#) > **stochasticProcess**

## 7.596 MultiAssetOption::results Class Reference

```
#include <ql/Instruments/multiassetoption.hpp>
```

Inheritance diagram for MultiAssetOption::results:



### 7.596.1 Detailed Description

Results from multi-asset option calculation

#### Public Member Functions

- void `reset()`

## 7.597 MultiCubicSpline Class Template Reference

```
#include <ql/Math/multicubicspline.hpp>
```

### 7.597.1 Detailed Description

```
template<Size i> class QuantLib::MultiCubicSpline< i >
```

#### Tests

interpolated values are checked against the original function.

#### Todo

- fix it for Borland compilation
- allow extrapolation as for the other interpolations
- investigate if and how to implement Hyman filters and different boundary conditions

#### Bug

- cannot interpolate at the grid points on the boundary surface of the N-dimensional region
- it does not compile under Borland

### Public Types

- typedef c\_splint::argument\_type **argument\_type**
- typedef c\_splint::result\_type **result\_type**
- typedef c\_splint::data\_table **data\_table**
- typedef c\_splint::return\_type **return\_type**
- typedef c\_splint::output\_data **output\_data**
- typedef c\_splint::dimensions **dimensions**
- typedef c\_splint::data **data**

### Public Member Functions

- **MultiCubicSpline** (const SplineGrid &grid, const data\_table &y, const std::vector< bool > &ae=std::vector< bool >(20, false))
- result\_type **operator()** (const argument\_type &x) const
- void **set\_shared\_increments** () const
- void **set\_shared\_coefficients** (const argument\_type &x) const

## 7.598 MultiPath Class Reference

```
#include <ql/MonteCarlo/multipath.hpp>
```

### 7.598.1 Detailed Description

Correlated multiple asset paths.

[MultiPath](#) contains the list of paths for each asset, i.e., `multipath[j]` is the path followed by the *j*-th asset.

### 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
- const [Path](#) & **at** (Size j) const
- [Path](#) & **operator**[ ] (Size j)
- [Path](#) & **at** (Size j)

## 7.599 MultiPathGenerator Class Template Reference

```
#include <ql/MonteCarlo/multipathgenerator.hpp>
```

### 7.599.1 Detailed Description

```
template<class GSG> class QuantLib::MultiPathGenerator< GSG >
```

Generates a multipath from a random number generator.

RSG is a sample generator which returns a random sequence. It must have the minimal interface:

```
RSG {  
    Sample<Array> next();  
};
```

#### Tests

the generated paths are checked against cached results

### Public Types

- typedef [Sample](#)< [MultiPath](#) > [sample\\_type](#)

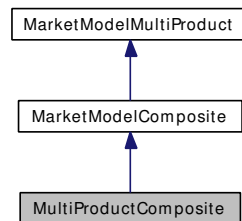
### Public Member Functions

- **MultiPathGenerator** (const boost::shared\_ptr< [StochasticProcess](#) > &, const [TimeGrid](#) &, GSG generator, bool brownianBridge=false)
- const [sample\\_type](#) & **next** () const
- const [sample\\_type](#) & **antithetic** () const

## 7.600 MultiProductComposite Class Reference

```
#include <ql/MarketModels/Products/multiproductcomposite.hpp>
```

Inheritance diagram for MultiProductComposite:



### 7.600.1 Detailed Description

Composition of one or more market-model products.

Instances of this class build a multiple market-model product by composing two or more sub-products.

#### Precondition:

All subproducts must have the same rate times.

### Public Member Functions

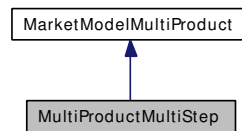
#### MarketModelMultiProduct interface

- Size **numberOfProducts** () const
- Size **maxNumberOfCashFlowsPerProductPerStep** () const
- bool **nextTimeStep** (const [CurveState](#) &currentState, std::vector< Size > &numberCashFlowsThisStep, std::vector< std::vector< [CashFlow](#) > > &cashFlowsGenerated)
- std::auto\_ptr< [MarketModelMultiProduct](#) > **clone** () const  
*returns a newly-allocated copy of itself*

## 7.601 MultiProductMultiStep Class Reference

```
#include <ql/MarketModels/Products/multiproductmultistep.hpp>
```

Inheritance diagram for MultiProductMultiStep:



### 7.601.1 Detailed Description

This is the abstract base class that encapsulates the notion of a [MarketModelMultiProduct](#) which can be evaluated in a more than one step (aka Rebonato's long jump).

#### Public Member Functions

- **MultiProductMultiStep** (const std::vector< Time > &rateTimes)

#### MarketModelMultiProduct interface

- std::vector< Size > **suggestedNumeraires** () const
- const [EvolutionDescription](#) & **evolution** () const

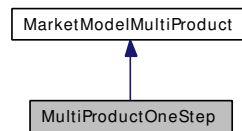
#### Protected Attributes

- std::vector< Time > **rateTimes\_**
- [EvolutionDescription](#) **evolution\_**

## 7.602 MultiProductOneStep Class Reference

```
#include <ql/MarketModels/Products/multiproductonestep.hpp>
```

Inheritance diagram for MultiProductOneStep:



### 7.602.1 Detailed Description

This is the abstract base class that encapsulates the notion of a [MarketModelMultiProduct](#) which can be evaluated in one step (aka Rebonato's very long jump).

#### Public Member Functions

- **MultiProductOneStep** (const std::vector< Time > &rateTimes)

#### MarketModelMultiProduct interface

- const [EvolutionDescription](#) & **evolution** () const
- std::vector< Size > **suggestedNumeraires** () const

#### Protected Attributes

- std::vector< Time > **rateTimes\_**
- [EvolutionDescription](#) **evolution\_**



## 7.603 MultiVariate Struct Template Reference

```
#include <ql/MonteCarlo/mctraits.hpp>
```

### 7.603.1 Detailed Description

**template<class RNG = PseudoRandom> struct QuantLib::MultiVariate< RNG >**

default Monte Carlo traits for multi-variate models

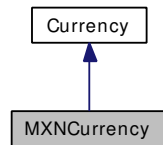
#### Public Types

- enum { **allowsErrorEstimate** = RNG::allowsErrorEstimate }
- typedef RNG **rng\_traits**
- typedef [MultiPath](#) **path\_type**
- typedef [PathPricer](#)< [path\\_type](#) > **path\_pricer\_type**
- typedef RNG::rsg\_type **rsg\_type**
- typedef [MultiPathGenerator](#)< rsg\_type > **path\_generator\_type**

## 7.604 MXNCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for MXNCurrency:



### 7.604.1 Detailed Description

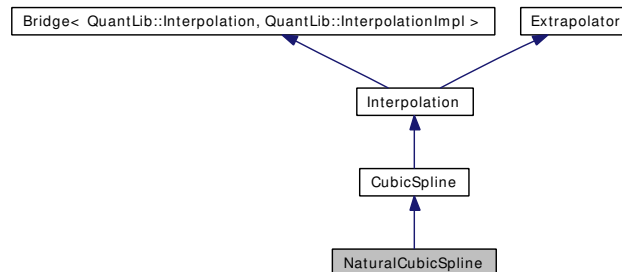
Mexican peso.

The ISO three-letter code is MXN; the numeric code is 484. It is divided in 100 centavos.

## 7.605 NaturalCubicSpline Class Reference

```
#include <ql/Math/cubicspline.hpp>
```

Inheritance diagram for NaturalCubicSpline:



### 7.605.1 Detailed Description

Cubic spline with null second derivative at end points

#### Public Member Functions

- `template<class I1, class I2> NaturalCubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)`

### 7.605.2 Constructor & Destructor Documentation

7.605.2.1 [NaturalCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

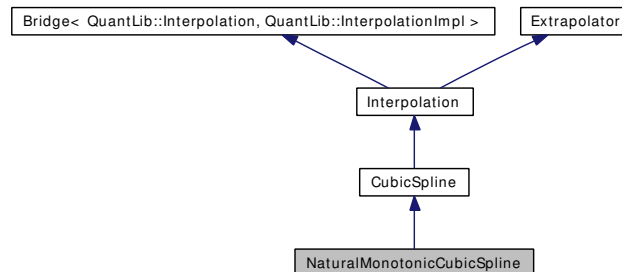
**Precondition:**

the *x* values must be sorted.

## 7.606 NaturalMonotonicCubicSpline Class Reference

```
#include <ql/Math/cubicspline.hpp>
```

Inheritance diagram for NaturalMonotonicCubicSpline:



### 7.606.1 Detailed Description

Natural cubic spline with monotonicity constraint.

#### Public Member Functions

- `template<class I1, class I2> NaturalMonotonicCubicSpline (const I1 &xBegin, const I1 &x-End, const I2 &yBegin)`

### 7.606.2 Constructor & Destructor Documentation

7.606.2.1 [NaturalMonotonicCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

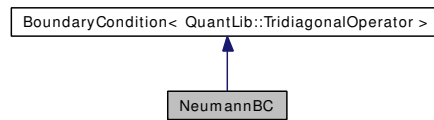
**Precondition:**

the *x* values must be sorted.

## 7.607 NeumannBC Class Reference

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Inheritance diagram for NeumannBC:



### 7.607.1 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** (Real value, [Side](#) side)
- void **applyBeforeApplying** ([TridiagonalOperator](#) &) const
- void **applyAfterApplying** ([Array](#) &) const
- void **applyBeforeSolving** ([TridiagonalOperator](#) &, [Array](#) &rhs) const
- void **applyAfterSolving** ([Array](#) &) const
- void **setTime** (Time)

### 7.607.2 Member Function Documentation

#### 7.607.2.1 void setTime (Time) [virtual]

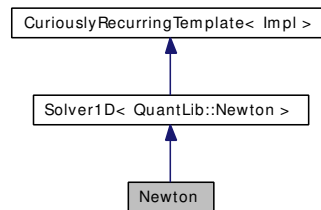
This method sets the current time for time-dependent boundary conditions.

Implements [BoundaryCondition](#).

## 7.608 Newton Class Reference

```
#include <ql/Solvers1D/newton.hpp>
```

Inheritance diagram for Newton:



### 7.608.1 Detailed Description

Newton 1-D solver

**Note:**

This solver requires that the passed function object implement a method `Real derivative(Real)`.

**Tests**

the correctness of the returned values is tested by checking them against known good results.

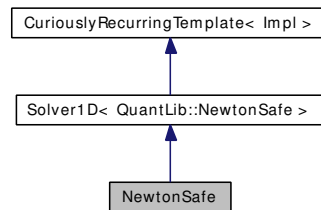
### Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

## 7.609 NewtonSafe Class Reference

```
#include <ql/Solvers1D/newtonsafe.hpp>
```

Inheritance diagram for NewtonSafe:



### 7.609.1 Detailed Description

safe Newton 1-D solver

#### Note:

This solver requires that the passed function object implement a method `Real derivative(Real)`.

#### Tests

the correctness of the returned values is tested by checking them against known good results.

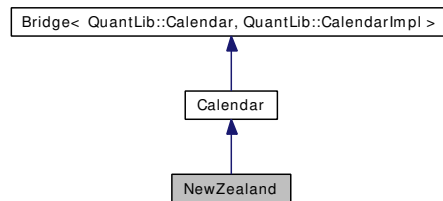
### Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

## 7.610 NewZealand Class Reference

```
#include <ql/Calendars/newzealand.hpp>
```

Inheritance diagram for NewZealand:



### 7.610.1 Detailed Description

New Zealand 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:**

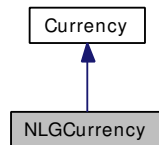
The holiday rules for New Zealand were documented by David Gilbert for IDB (<http://www.jrefinery.com/ibd/>)



## 7.611 NLGCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for NLGCurrency:



### 7.611.1 Detailed Description

Dutch guilder.

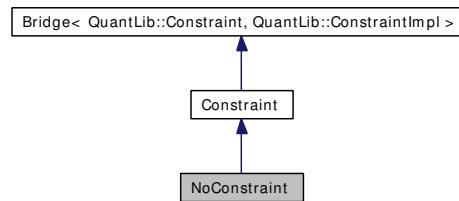
The ISO three-letter code was NLG; the numeric code was 528. It was divided in 100 cents.

Obsoleted by the Euro since 1999.

## 7.612 NoConstraint Class Reference

```
#include <ql/Optimization/constraint.hpp>
```

Inheritance diagram for NoConstraint:



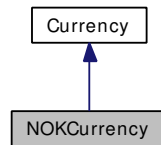
### 7.612.1 Detailed Description

No constraint.

## 7.613 NOKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for NOKCurrency:



### 7.613.1 Detailed Description

Norwegian krone.

The ISO three-letter code is NOK; the numeric code is 578. It is divided in 100 øre.

## 7.614 NonLinearLeastSquare Class Reference

```
#include <ql/Optimization/leastsquare.hpp>
```

### 7.614.1 Detailed Description

Non-linear least-square method.

Using a given optimization algorithm (default is conjugate gradient),

$$\min\{r(x) : x \in R^n\}$$

where  $r(x) = |f(x)|^2$  is the Euclidean norm of  $f(x)$  for some vector-valued function  $f$  from  $R^n$  to  $R^m$ ,

$$f = (f_1, \dots, f_m)$$

with  $f_i(x) = b_i - \phi(x, t_i)$  where  $b$  is the vector of target data and  $\phi$  is a scalar function.

Assuming the differentiability of  $f$ , the gradient of  $r$  is defined by

$$\text{grad}r(x) = f'(x)^t \cdot f(x)$$

### Public Member Functions

- [NonLinearLeastSquare](#) ([Constraint](#) &c, Real accuracy=1e-4, Size maxiter=100)  
*Default constructor.*
- [NonLinearLeastSquare](#) ([Constraint](#) &c, Real accuracy, Size maxiter, boost::shared\_ptr<[OptimizationMethod](#)> 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*
- Real [residualNorm](#) ()  
*return the least square residual norm*
- Real [lastValue](#) ()  
*return last function value*
- Integer [exitFlag](#) ()  
*return exit flag*
- Integer [iterationsNumber](#) ()  
*return the performed number of iterations*

## 7.615 NormalDistribution Class Reference

```
#include <ql/Math/normaldistribution.hpp>
```

### 7.615.1 Detailed Description

Normal distribution function.

Given  $x$ , it returns its probability in a Gaussian normal distribution. It provides the first derivative too.

#### Tests

the correctness of the returned value is tested by checking it against numerical calculations. Cross-checks are also performed against the [CumulativeNormalDistribution](#) and [InverseCumulativeNormal](#) classes.

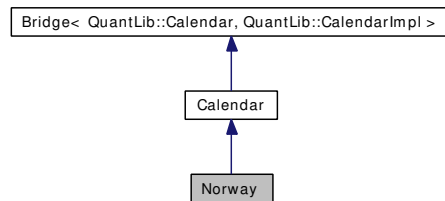
### Public Member Functions

- **NormalDistribution** (Real average=0.0, Real sigma=1.0)
- Real **operator()** (Real  $x$ ) const
- Real **derivative** (Real  $x$ ) const

## 7.616 Norway Class Reference

```
#include <ql/Calendars/norway.hpp>
```

Inheritance diagram for Norway:



### 7.616.1 Detailed Description

Norwegian calendar.

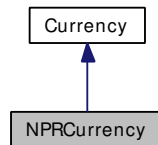
Holidays:

- 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

## 7.617 NPRCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for NPRCurrency:



### 7.617.1 Detailed Description

Nepal rupee.

The ISO three-letter code is NPR; the numeric code is 524. It is divided in 100 paise.

## 7.618 Null Class Template Reference

```
#include <ql/Utilities/null.hpp>
```

### 7.618.1 Detailed Description

```
template<class Type> class QuantLib::Null< Type >
```

template class providing a null value for a given type.

### Public Member Functions

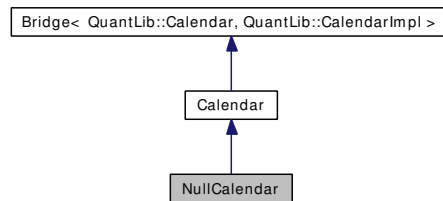
- `operator Type () const`



## 7.619 NullCalendar Class Reference

```
#include <ql/Calendars/nullcalendar.hpp>
```

Inheritance diagram for NullCalendar:



### 7.619.1 Detailed Description

Calendar for reproducing theoretical calculations.

This calendar has no holidays. It ensures that dates at whole-month distances have the same day of month.

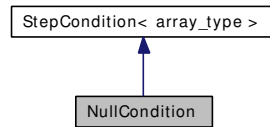
**Examples:**

[Replication.cpp](#), and [Repo.cpp](#).

## 7.620 NullCondition Class Template Reference

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

Inheritance diagram for NullCondition:



### 7.620.1 Detailed Description

```
template<class array_type> class QuantLib::NullCondition< array_type >
```

null step condition

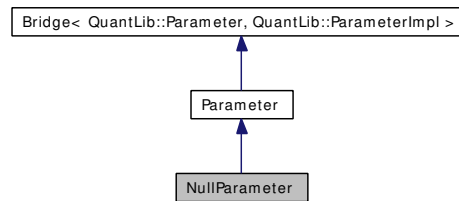
#### Public Member Functions

- void **applyTo** (array\_type &, Time) const

## 7.621 NullParameter Class Reference

```
#include <ql/ShortRateModels/parameter.hpp>
```

Inheritance diagram for NullParameter:



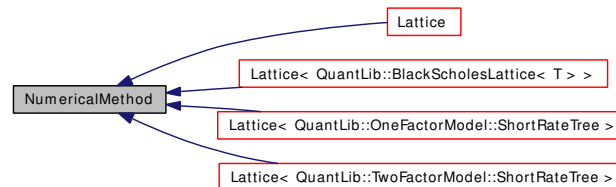
### 7.621.1 Detailed Description

Parameter which is always zero  $a(t) = 0$

## 7.622 NumericalMethod Class Reference

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

Inheritance diagram for NumericalMethod:



### 7.622.1 Detailed Description

Numerical method (tree, finite-differences) base class.

#### Public Member Functions

- **NumericalMethod** (const [TimeGrid](#) &timeGrid)
- virtual [Disposable](#)< [Array](#) > **grid** (Time) const=0

#### Inspectors

- const [TimeGrid](#) & **timeGrid** () const

#### Numerical method interface

*These methods are to be used by discretized assets and must be overridden by developers implementing numerical methods. Users are advised to use the corresponding methods of [DiscretizedAsset](#) instead.*

- virtual void **initialize** ([DiscretizedAsset](#) &, Time time) const =0  
*initialize an asset at the given time.*
- virtual void **rollback** ([DiscretizedAsset](#) &, Time to) const=0
- virtual void **partialRollback** ([DiscretizedAsset](#) &, Time to) const=0
- virtual [Real](#) **presentValue** ([DiscretizedAsset](#) &) const=0  
*computes the present value of an asset.*

#### Protected Attributes

- [TimeGrid](#) t\_

### 7.622.2 Member Function Documentation

#### 7.622.2.1 virtual void rollback ([DiscretizedAsset](#) &, Time to) const [pure virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implemented in [Lattice](#), [TsiveriotisFernandesLattice](#), [Lattice< QuantLib::BlackScholesLattice< T > >](#), [Lattice< QuantLib::OneFactorModel::ShortRateTree >](#), and [Lattice< QuantLib::TwoFactorModel::ShortRateTree >](#).

**7.622.2.2 virtual void partialRollback ([DiscretizedAsset](#) &, Time *to*) const** [pure virtual]

Roll back an asset until the given time, but do not perform the final adjustment.

**Warning**

In version 0.3.7 and earlier, this method was called `rollAlmostBack` method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

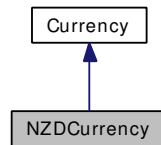
```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Implemented in [Lattice](#), [TsiveriotisFernandesLattice](#), [Lattice< QuantLib::BlackScholesLattice< T > >](#), [Lattice< QuantLib::OneFactorModel::ShortRateTree >](#), and [Lattice< QuantLib::TwoFactorModel::ShortRateTree >](#).

## 7.623 NZDCurrency Class Reference

```
#include <ql/Currencies/oceania.hpp>
```

Inheritance diagram for NZDCurrency:



### 7.623.1 Detailed Description

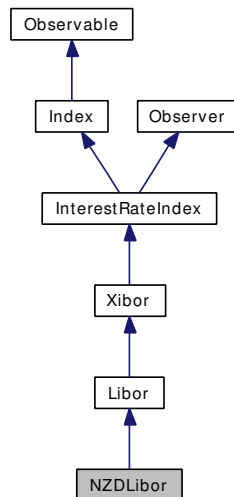
New Zealand dollar.

The ISO three-letter code is NZD; the numeric code is 554. It is divided in 100 cents.

## 7.624 NZDLibor Class Reference

```
#include <ql/Indexes/nzdlibor.hpp>
```

Inheritance diagram for NZDLibor:



### 7.624.1 Detailed Description

NZD LIBOR rate

New Zealand Dollar LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

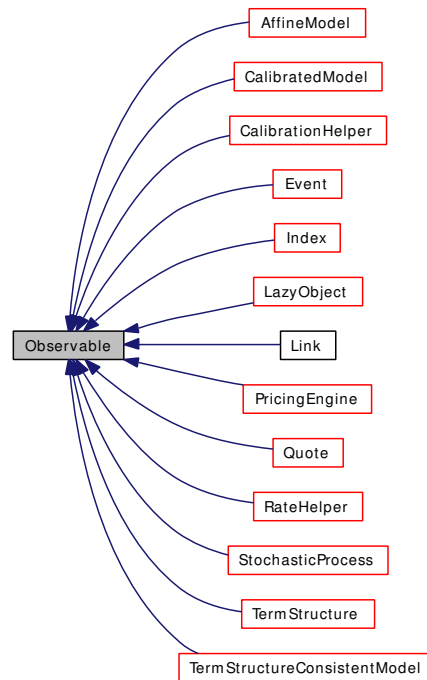
### Public Member Functions

- **NZDLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference, [Integer](#) settlementDays=2)

## 7.625 Observable Class Reference

```
#include <ql/Patterns/observable.hpp>
```

Inheritance diagram for Observable:



### 7.625.1 Detailed Description

Object that notifies its changes to a set of observables.

#### Public Member Functions

- `Observable` (const `Observable` &)
- `Observable` & `operator=` (const `Observable` &)
- void `notifyObservers` ()

#### Friends

- class `Observer`



## 7.625.2 Member Function Documentation

### 7.625.2.1 **Observable** & operator= (const **Observable** & o)

#### **Warning**

notification is sent before the copy constructor has a chance of actually change the data members. Therefore, observers whose update() method tries to use their observables will not see the updated values. It is suggested that the update() method just raise a flag in order to trigger a later recalculation.

### 7.625.2.2 void notifyObservers ()

This method should be called at the end of non-const methods or when the programmer desires to notify any changes.

## 7.626 ObservableValue Class Template Reference

```
#include <ql/Utilities/observablevalue.hpp>
```

### 7.626.1 Detailed Description

**template<class T> class QuantLib::ObservableValue< T >**

observable and assignable proxy to concrete value

Observers can be registered with instances of this class so that they are notified when a different value is assigned to such instances. Client code can copy the contained value or pass it to functions via implicit conversion.

#### Note:

it is not possible to call non-const method on the returned value. This is by design, as this possibility would necessarily bypass the notification code; client code should modify the value via re-assignment instead.

### Public Member Functions

- **ObservableValue** (const T &)
- **ObservableValue** (const [ObservableValue](#)< T > &)
- **operator T** () const  
*implicit conversion*
- **operator boost::shared\_ptr** () const
- const T & **value** () const  
*explicit inspector*

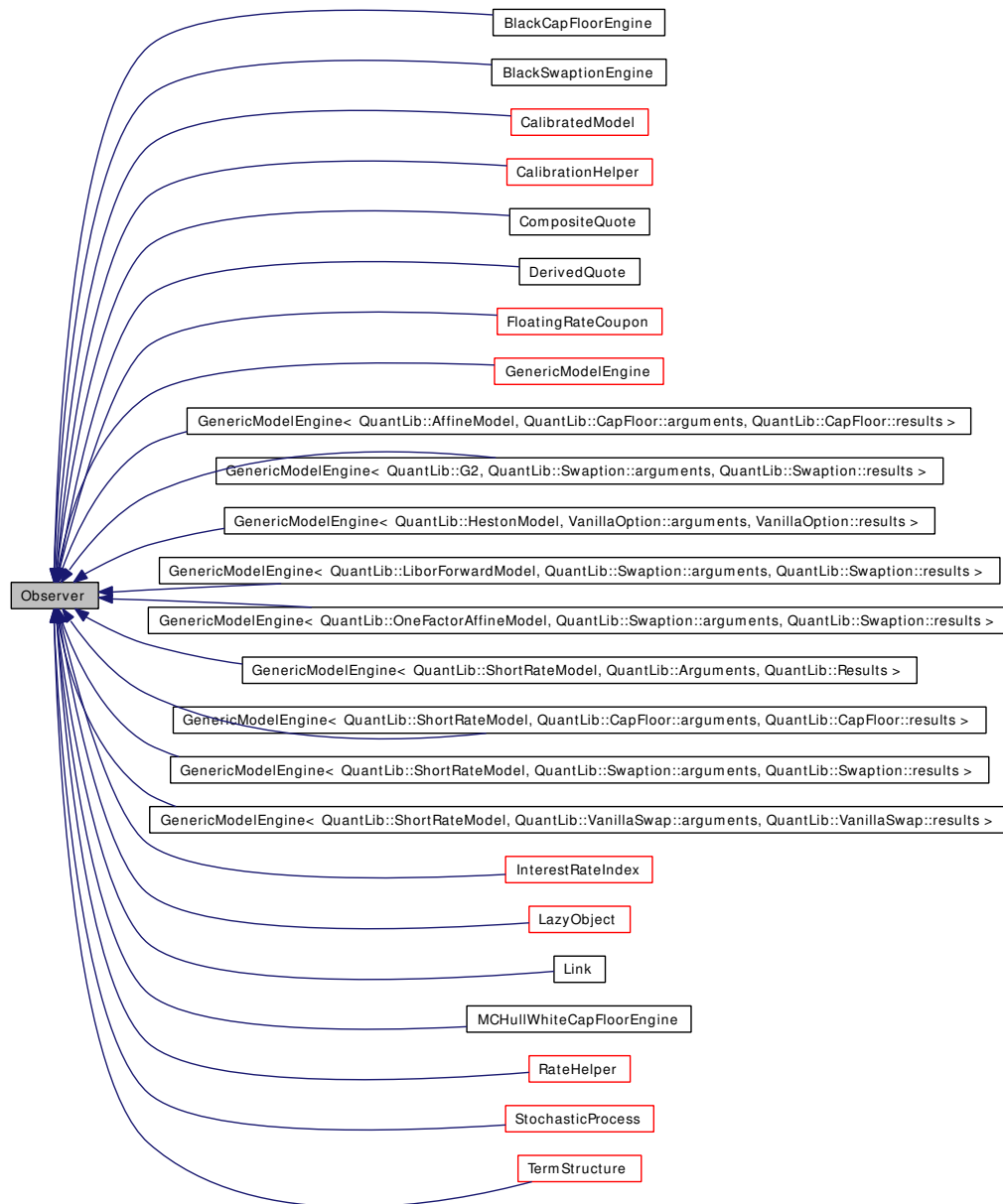
#### controlled assignment

- [ObservableValue](#)< T > & **operator=** (const T &)
- [ObservableValue](#)< T > & **operator=** (const [ObservableValue](#)< T > &)

## 7.627 Observer Class Reference

```
#include <ql/Patterns/observable.hpp>
```

Inheritance diagram for Observer:



### 7.627.1 Detailed Description

Object that gets notified when a given observable changes.

## Public Member Functions

- **Observer** (const [Observer](#) &)
- **Observer** & **operator=** (const [Observer](#) &)
- void **registerWith** (const boost::shared\_ptr< [Observable](#) > &)
- void **unregisterWith** (const boost::shared\_ptr< [Observable](#) > &)
- virtual void **update** ()=0

## 7.627.2 Member Function Documentation

### 7.627.2.1 virtual void update () [pure virtual]

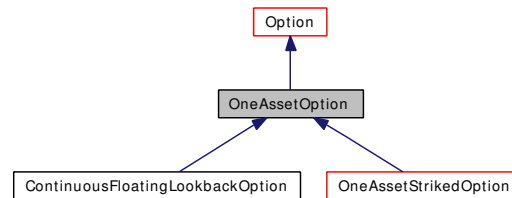
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implemented in [FloatingRateCoupon](#), [Link](#), [InterestRateIndex](#), [LazyObject](#), [BlackCapFloorEngine](#), [MCHullWhiteCapFloorEngine](#), [GenericModelEngine](#), [LatticeShortRateModelEngine](#), [BlackSwaptionEngine](#), [GeneralizedBlackScholesProcess](#), [DerivedQuote](#), [CompositeQuote](#), [CalibrationHelper](#), [CalibratedModel](#), [StochasticProcess](#), [TermStructure](#), [ExtendedDiscountCurve](#), [FlatForward](#), [RateHelper](#), [PiecewiseYieldCurve](#), [PiecewiseZeroSpreadedTermStructure](#), [RelativeDateRateHelper](#), [CapVolatilityVector](#), [GenericModelEngine< QuantLib::ShortRateModel, QuantLib::VanillaSwap::arguments, QuantLib::VanillaSwap::results >](#), [GenericModelEngine< QuantLib::OneFactorAffineModel, QuantLib::Swaption::arguments, QuantLib::Swaption::results >](#), [GenericModelEngine< QuantLib::G2, QuantLib::Swaption::arguments, QuantLib::Swaption::results >](#), [GenericModelEngine< QuantLib::ShortRateModel, QuantLib::CapFloor::arguments, QuantLib::CapFloor::results >](#), [GenericModelEngine< QuantLib::LiborForwardModel, QuantLib::Swaption::arguments, QuantLib::Swaption::results >](#), [GenericModelEngine< QuantLib::ShortRateModel, QuantLib::Arguments, QuantLib::Results >](#), [GenericModelEngine< QuantLib::ShortRateModel, QuantLib::Swaption::arguments, QuantLib::Swaption::results >](#), [GenericModelEngine< QuantLib::AffineModel, QuantLib::CapFloor::arguments, QuantLib::CapFloor::results >](#), [GenericModelEngine< QuantLib::HestonModel, VanillaOption::arguments, VanillaOption::results >](#), [LatticeShortRateModelEngine< QuantLib::VanillaSwap::arguments, QuantLib::VanillaSwap::results >](#), [LatticeShortRateModelEngine< QuantLib::Swaption::arguments, QuantLib::Swaption::results >](#), and [LatticeShortRateModelEngine< QuantLib::CapFloor::arguments, QuantLib::CapFloor::results >](#).

## 7.628 OneAssetOption Class Reference

```
#include <ql/Instruments/oneassetoption.hpp>
```

Inheritance diagram for OneAssetOption:



### 7.628.1 Detailed Description

Base class for options on a single asset.

#### Public Member Functions

- **OneAssetOption** (const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [Payoff](#) > &, const boost::shared\_ptr< [Exercise](#) > &, const boost::shared\_ptr< [PricingEngine](#) > & engine=boost::shared\_ptr< [PricingEngine](#) >())
- **Volatility impliedVolatility** (Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility minVol=QL\_MIN\_VOLATILITY, Volatility maxVol=QL\_MAX\_VOLATILITY) const
- void **setupArguments** ([Arguments](#) \*) const
- void **fetchResults** (const [Results](#) \*) const

#### Instrument interface

- bool **isExpired** () const  
*returns whether the instrument is still tradable.*

#### greeks

- Real **delta** () const
- Real **deltaForward** () const
- Real **elasticity** () const
- Real **gamma** () const
- Real **theta** () const
- Real **thetaPerDay** () const
- Real **vega** () const
- Real **rho** () const
- Real **dividendRho** () const
- Real **itmCashProbability** () const
- [SampledCurve](#) **priceCurve** () const

#### Protected Member Functions

- void **setupExpired** () const

## Protected Attributes

- Real `delta_`
- Real `deltaForward_`
- Real `elasticity_`
- Real `gamma_`
- Real `theta_`
- Real `thetaPerDay_`
- Real `vega_`
- Real `rho_`
- Real `dividendRho_`
- Real `itmCashProbability_`
- [SampledCurve](#) `priceCurve_`
- `boost::shared_ptr< StochasticProcess >` `stochasticProcess_`

## Classes

- class [arguments](#)  
*Arguments for single-asset option calculation*
- class [results](#)  
*Results from single-asset option calculation*

## 7.628.2 Member Function Documentation

- 7.628.2.1 [Volatility](#) `impliedVolatility (Real price, Real accuracy = 1.0e-4, Size maxEvaluations = 100, Volatility minVol = QL_MIN_VOLATILITY, Volatility maxVol = QL_MAX_VOLATILITY) const`

### Warning

currently, this method returns the Black-Scholes implied volatility. It will give inconsistent results if the pricing was performed with any other methods (such as jump-diffusion models.)

### Warning

options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g. binary options. In these cases the calculation can fail and the result (if any) is almost meaningless. Another possible source of failure is to have a target value that is not attainable with any volatility, e.g., a target value lower than the intrinsic value in the case of American options.

- 7.628.2.2 `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](#).

Reimplemented in [ContinuousAveragingAsianOption](#), [DiscreteAveragingAsianOption](#), [BarrierOption](#), [CliquetOption](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [ContinuousFloatingLookbackOption](#), [ContinuousFixedLookbackOption](#), [OneAssetStrikedOption](#), [QuantoForwardVanillaOption](#), and [QuantoVanillaOption](#).

#### 7.628.2.3 void fetchResults (const [Results](#) \*) const [virtual]

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

Reimplemented from [Instrument](#).

Reimplemented in [ForwardVanillaOption](#), [OneAssetStrikedOption](#), and [QuantoVanillaOption](#).

#### 7.628.2.4 void setupExpired () const [protected, virtual]

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

Reimplemented from [Instrument](#).

Reimplemented in [OneAssetStrikedOption](#), and [QuantoVanillaOption](#).

## 7.629 OneAssetOption::arguments Class Reference

```
#include <ql/Instruments/oneassetoption.hpp>
```

### 7.629.1 Detailed Description

Arguments for single-asset option calculation

#### Public Member Functions

- void **validate** () const

#### Public Attributes

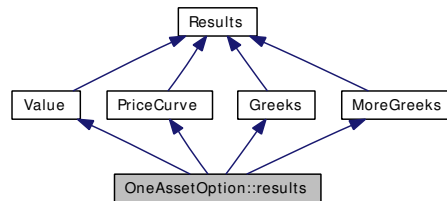
- boost::shared\_ptr< [StochasticProcess](#) > **stochasticProcess**



## 7.630 OneAssetOption::results Class Reference

```
#include <ql/Instruments/oneassetoption.hpp>
```

Inheritance diagram for OneAssetOption::results:



### 7.630.1 Detailed Description

Results from single-asset option calculation

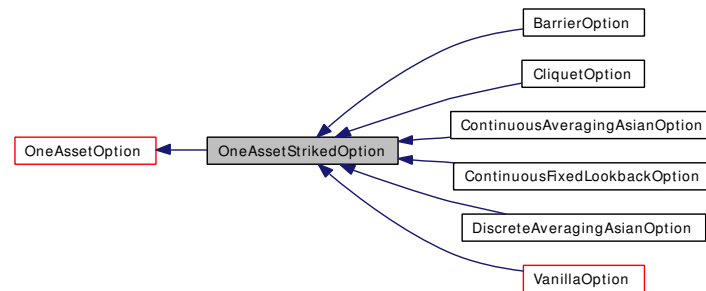
#### Public Member Functions

- `void reset ()`

## 7.631 OneAssetStrikedOption Class Reference

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Inheritance diagram for OneAssetStrikedOption:



### 7.631.1 Detailed Description

Base class for options on a single asset with striked payoff.

#### Public Member Functions

- **OneAssetStrikedOption** (const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &, const boost::shared\_ptr< [Exercise](#) > &, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) \*) const
- void [fetchResults](#) (const [Results](#) \*) const

#### greeks

- Real [strikeSensitivity](#) () const

#### Protected Member Functions

- void [setupExpired](#) () const

#### Protected Attributes

- Real [strikeSensitivity\\_](#)

### 7.631.2 Member Function Documentation

#### 7.631.2.1 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 [OneAssetOption](#).

Reimplemented in [ContinuousAveragingAsianOption](#), [DiscreteAveragingAsianOption](#), [BarrierOption](#), [CliquetOption](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [ContinuousFixed-LookbackOption](#), [QuantoForwardVanillaOption](#), and [QuantoVanillaOption](#).

#### 7.631.2.2 void fetchResults (const [Results](#) \*) const [virtual]

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

Reimplemented from [OneAssetOption](#).

Reimplemented in [ForwardVanillaOption](#), and [QuantoVanillaOption](#).

#### 7.631.2.3 void setupExpired () const [protected, virtual]

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

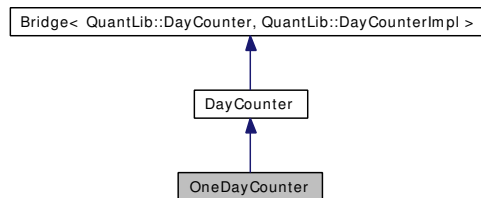
Reimplemented from [OneAssetOption](#).

Reimplemented in [QuantoVanillaOption](#).

## 7.632 OneDayCounter Class Reference

```
#include <ql/DayCounters/one.hpp>
```

Inheritance diagram for OneDayCounter:



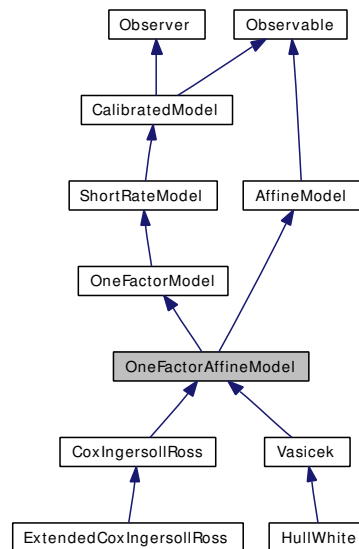
### 7.632.1 Detailed Description

1/1 day count convention

## 7.633 OneFactorAffineModel Class Reference

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Inheritance diagram for OneFactorAffineModel:



### 7.633.1 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)
- virtual **Real discountBond** (Time now, Time maturity, [Array](#) factors) const
- **Real discountBond** (Time now, Time maturity, Rate rate) const
- **DiscountFactor discount** (Time t) const

*Implied discount curve.*

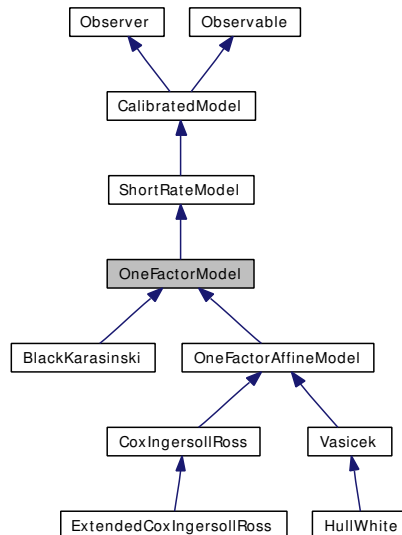
### Protected Member Functions

- virtual **Real A** (Time t, Time T) const=0
- virtual **Real B** (Time t, Time T) const=0

## 7.634 OneFactorModel Class Reference

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Inheritance diagram for OneFactorModel:



### 7.634.1 Detailed Description

Single-factor short-rate model abstract class.

#### Public Member Functions

- **OneFactorModel** (Size nArguments)
- virtual `boost::shared_ptr< ShortRateDynamics > dynamics () const=0`  
*returns the short-rate dynamics*
- `boost::shared_ptr< NumericalMethod > tree (const TimeGrid &grid) const`  
*Return by default a trinomial recombining tree.*

#### Classes

- class `ShortRateDynamics`  
*Base class describing the short-rate dynamics.*
- class `ShortRateTree`  
*Recombining trinomial tree discretizing the state variable.*

## 7.635 OneFactorModel::ShortRateDynamics Class Reference

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

### 7.635.1 Detailed Description

Base class describing the short-rate dynamics.

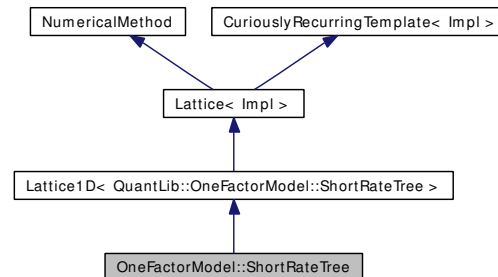
#### Public Member Functions

- **ShortRateDynamics** (const boost::shared\_ptr< [StochasticProcess1D](#) > &process)
- virtual [Real](#) [variable](#) ([Time](#) t, [Rate](#) r) const=0  
*Compute state variable from short rate.*
- virtual [Rate](#) [shortRate](#) ([Time](#) t, [Real](#) variable) const=0  
*Compute short rate from state variable.*
- const boost::shared\_ptr< [StochasticProcess1D](#) > & [process](#) ()  
*Returns the risk-neutral dynamics of the state variable.*

## 7.636 OneFactorModel::ShortRateTree Class Reference

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Inheritance diagram for OneFactorModel::ShortRateTree:



### 7.636.1 Detailed Description

Recombining trinomial tree discretizing the state variable.

#### Public Member Functions

- [ShortRateTree](#) (const boost::shared\_ptr< [TrinomialTree](#) > &tree, const boost::shared\_ptr< [ShortRateDynamics](#) > &dynamics, const [TimeGrid](#) &timeGrid)  
*Plain tree build-up from short-rate dynamics.*
- [ShortRateTree](#) (const boost::shared\_ptr< [TrinomialTree](#) > &tree, const boost::shared\_ptr< [ShortRateDynamics](#) > &dynamics, const boost::shared\_ptr< [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
- [Real](#) **underlying** (Size i, Size index) const
- Size **descendant** (Size i, Size index, Size branch) const
- [Real](#) **probability** (Size i, Size index, Size branch) const



## 7.637 OperatorFactory Class Reference

```
#include <ql/FiniteDifferences/operatorfactory.hpp>
```

### 7.637.1 Detailed Description

Black-Scholes-Merton differential operator.

#### Tests

coefficients are tested against constant BSM operator

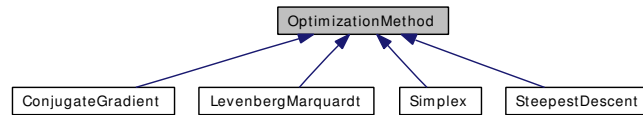
### Static Public Member Functions

- static [TridiagonalOperator](#) **getOperator** (const boost::shared\_ptr< [GeneralizedBlackScholesProcess](#) > &process, const [Array](#) &grid, [Time](#) residualTime, bool timeDependent)
- static [TridiagonalOperator](#) **getOperator** (const boost::shared\_ptr< [OneFactorModel::ShortRateDynamics](#) > &process, const [Array](#) &grid)

## 7.638 OptimizationMethod Class Reference

```
#include <ql/Optimization/method.hpp>
```

Inheritance diagram for OptimizationMethod:



### 7.638.1 Detailed Description

Abstract class for constrained optimization method.

#### Public Member Functions

- **OptimizationMethod** (const [EndCriteria](#) &endCriteria, const [Array](#) &initialValue)
- void [setInitialValue](#) (const [Array](#) &initialValue)  
*Set initial value.*
- void [setEndCriteria](#) (const [EndCriteria](#) &endCriteria)  
*Set optimization end criteria.*
- Integer & [iterationNumber](#) () const  
*current iteration number*
- [EndCriteria](#) & [endCriteria](#) () const  
*optimization end criteria*
- Integer & [functionEvaluation](#) () const  
*number of evaluation of cost function*
- Integer & [gradientEvaluation](#) () const  
*number of evaluation of cost function gradient*
- Real & [functionValue](#) () const  
*value of cost function*
- Real & [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  $P$*

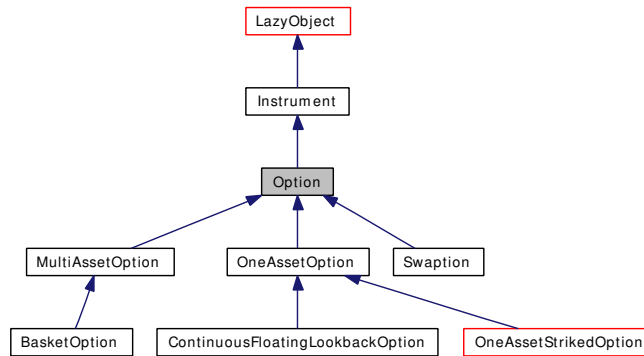
## Protected Attributes

- Integer `iterationNumber_`  
*current iteration step in the Optimization process*
- Integer `functionEvaluation_`  
*number of evaluation of cost function and its gradient*
- Integer `gradientEvaluation_`
- Real `functionValue_`  
*function and gradient norm values of the last step*
- Real `squaredNorm_`
- `EndCriteria` `endCriteria_`  
*optimization end criteria*
- Array `initialValue_`  
*initial value of unknowns*
- Array `x_`  
*current values of the local minimum and the search direction*
- Array `searchDirection_`

## 7.639 Option Class Reference

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

Inheritance diagram for Option:



### 7.639.1 Detailed Description

base option class

#### Public Types

- enum **Type** { Put = -1, Call = 1 }

#### Public Member Functions

- Option** (const boost::shared\_ptr< [Payoff](#) > &payoff, const boost::shared\_ptr< [Exercise](#) > &exercise, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())

#### Protected Attributes

- boost::shared\_ptr< [Payoff](#) > **payoff\_**
- boost::shared\_ptr< [Exercise](#) > **exercise\_**

#### Related Functions

(Note that these are not member functions.)

- std::ostream & **operator<<** (std::ostream &, Option::Type)

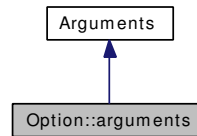
#### Classes

- class [arguments](#)

## 7.640 Option::arguments Class Reference

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

Inheritance diagram for Option::arguments:



### 7.640.1 Detailed Description

basic option arguments

#### Todo

- remove `std::vector<Time> stoppingTimes`
- how to handle strike-less option (asian average strike, forward, etc.)?

### Public Member Functions

- void **validate** () const

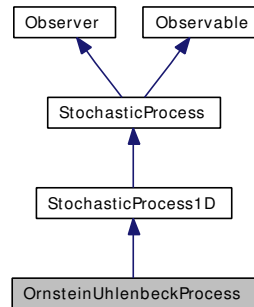
### Public Attributes

- boost::shared\_ptr< [Payoff](#) > **payoff**
- boost::shared\_ptr< [Exercise](#) > **exercise**
- std::vector< Time > **stoppingTimes**

## 7.641 OrnsteinUhlenbeckProcess Class Reference

```
#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>
```

Inheritance diagram for OrnsteinUhlenbeckProcess:



### 7.641.1 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** (Real speed, Volatility vol, Real x0=0.0)

#### StochasticProcess interface

- Real **x0** () const  
*returns the initial value of the state variable*
- Real **drift** (Time t, Real x) const  
*returns the drift part of the equation, i.e.  $\mu(t, x_i)$*
- Real **diffusion** (Time t, Real x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_i)$*
- Real **expectation** (Time t0, Real x0, Time dt) const
- Real **stdDeviation** (Time t0, Real x0, Time dt) const
- Real **variance** (Time t0, Real x0, Time dt) const

### 7.641.2 Member Function Documentation

#### 7.641.2.1 Real expectation (Time t0, Real x0, Time dt) const [virtual]

returns the expectation  $E(x_{t_0+\Delta t} | x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

**7.641.2.2 Real stdDeviation (Time  $t0$ , Real  $x0$ , Time  $dt$ ) const [virtual]**

returns the standard deviation  $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

**7.641.2.3 Real variance (Time  $t0$ , Real  $x0$ , Time  $dt$ ) const [virtual]**

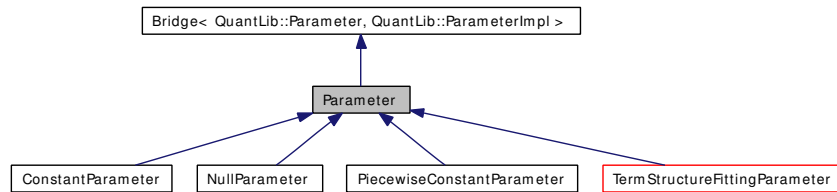
returns the variance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

## 7.642 Parameter Class Reference

#include <ql/ShortRateModels/parameter.hpp>

Inheritance diagram for Parameter:



### 7.642.1 Detailed Description

Base class for model arguments.

#### Public Member Functions

- const [Array](#) & **params** () const
- void **setParameter** (Size i, [Real](#) x)
- bool **testParams** (const [Array](#) &params) const
- Size **size** () const
- [Real](#) **operator()** ([Time](#) t) const
- const boost::shared\_ptr< [ParameterImpl](#) > & **implementation** () const

#### Protected Member Functions

- **Parameter** (Size size, const boost::shared\_ptr< [ParameterImpl](#) > &impl, const [Constraint](#) &constraint)

#### Protected Attributes

- [Array](#) **params\_**
- [Constraint](#) **constraint\_**



## 7.643 ParameterImpl Class Reference

```
#include <ql/ShortRateModels/parameter.hpp>
```

### 7.643.1 Detailed Description

Base class for model parameter implementation.

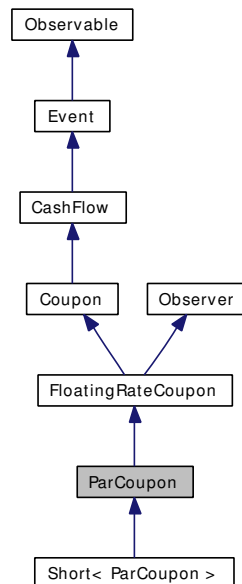
#### Public Member Functions

- virtual [Real](#) value (const [Array](#) &params, [Time](#) t) const=0

## 7.644 ParCoupon Class Reference

```
#include <ql/CashFlows/parcoupon.hpp>
```

Inheritance diagram for ParCoupon:



### 7.644.1 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** (const [Date](#) &paymentDate, const [Real](#) nominal, const [Date](#) &startDate, const [Date](#) &endDate, const [Integer](#) fixingDays, const boost::shared\_ptr< [Xibor](#) > &index, const [Real](#) gearing=1.0, const [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())

#### Coupon interface

- Rate [rate](#) () const  
*accrued rate*

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.645 Path Class Reference

```
#include <ql/MonteCarlo/path.hpp>
```

### 7.645.1 Detailed Description

single-factor random walk

**Note:**

the path includes the initial asset value as its first point.

**Examples:**

[DiscreteHedging.cpp](#).

### iterators

- typedef Array::const\_iterator **iterator**
- typedef Array::const\_reverse\_iterator **reverse\_iterator**
- iterator **begin** () const
- iterator **end** () const
- reverse\_iterator **rbegin** () const
- reverse\_iterator **rend** () const

### Public Member Functions

- Path (const [TimeGrid](#) &timeGrid, const [Array](#) &values=[Array](#)())

### inspectors

- bool **empty** () const
- Size **length** () const
- [Real](#) **operator[]** (Size i) const  
*asset value at the i-th point*
- [Real](#) **at** (Size i) const
- [Real](#) & **operator[]** (Size i)
- [Real](#) & **at** (Size i)
- [Real](#) **value** (Size i) const
- [Real](#) & **value** (Size i)
- [Time](#) **time** (Size i) const  
*time at the i-th point*
- [Real](#) **front** () const  
*initial asset value*
- [Real](#) & **front** ()
- [Real](#) **back** () const  
*final asset value*
- [Real](#) & **back** ()
- const [TimeGrid](#) & **timeGrid** () const  
*time grid*

## 7.646 PathGenerator Class Template Reference

```
#include <ql/MonteCarlo/pathgenerator.hpp>
```

### 7.646.1 Detailed Description

**template<class GSG> class QuantLib::PathGenerator< GSG >**

Generates random paths using a sequence generator.

Generates random paths with drift(S,t) and variance(S,t) using a gaussian sequence generator

#### Tests

the generated paths are checked against cached results

### Public Types

- typedef [Sample](#)< [Path](#) > **sample\_type**

### Public Member Functions

- **PathGenerator** (const boost::shared\_ptr< [StochasticProcess](#) > &, Time length, Size time-Steps, const GSG &generator, bool brownianBridge)
- **PathGenerator** (const boost::shared\_ptr< [StochasticProcess](#) > &, const [TimeGrid](#) &time-Grid, const GSG &generator, bool brownianBridge)

#### inspectors

- const [sample\\_type](#) & **next** () const
- const [sample\\_type](#) & **antithetic** () const
- Size **size** () const
- const [TimeGrid](#) & **timeGrid** () const

## 7.647 PathPricer Class Template Reference

```
#include <ql/MonteCarlo/pathpricer.hpp>
```

### 7.647.1 Detailed Description

```
template<class PathType, class ValueType = Real> class QuantLib::PathPricer< PathType,  
ValueType >
```

base class for path pricers

Returns the value of an option on a given path.

**Examples:**

[DiscreteHedging.cpp](#).

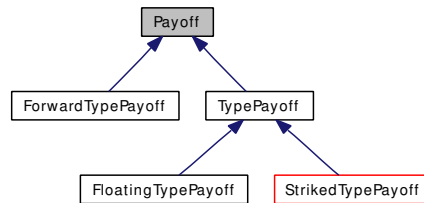
### Public Member Functions

- virtual ValueType **operator()** (const PathType &path) const =0

## 7.648 Payoff Class Reference

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

Inheritance diagram for Payoff:



### 7.648.1 Detailed Description

Base class for option payoffs.

#### Public Member Functions

##### Payoff interface

- virtual Real **operator()** (Real price) const=0

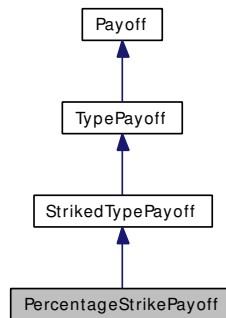
##### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.649 PercentageStrikePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for PercentageStrikePayoff:



### 7.649.1 Detailed Description

Payoff with strike expressed as percentage

#### Public Member Functions

- **PercentageStrikePayoff** (Option::Type type, Real moneyness)
- Real **operator()** (Real price) const
- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.650 Period Class Reference

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

### 7.650.1 Detailed Description

Time period described by a number of a given time unit.

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

### Public Member Functions

- **Period** (Integer n, [TimeUnit](#) units)
- **Period** ([Frequency](#) f)
- Integer **length** () const
- [TimeUnit](#) **units** () const
- [Frequency](#) **frequency** () const

### Related Functions

(Note that these are not member functions.)

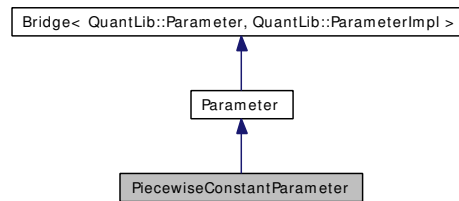
- [Period](#) **operator** \* (Integer n, [TimeUnit](#) units)
- [Period](#) **operator** \* ([TimeUnit](#) units, Integer n)
- [Period](#) **operator**- (const [Period](#) &)
- [Period](#) **operator** \* (Integer n, const [Period](#) &)
- [Period](#) **operator** \* (const [Period](#) &, Integer n)
- bool **operator**< (const [Period](#) &, const [Period](#) &)
- bool **operator**== (const [Period](#) &, const [Period](#) &)
- bool **operator**!= (const [Period](#) &, const [Period](#) &)
- bool **operator**> (const [Period](#) &, const [Period](#) &)
- bool **operator**<= (const [Period](#) &, const [Period](#) &)
- bool **operator**>= (const [Period](#) &, const [Period](#) &)
- std::ostream & **operator**<< (std::ostream &, const [Period](#) &)



## 7.651 PiecewiseConstantParameter Class Reference

```
#include <ql/ShortRateModels/parameter.hpp>
```

Inheritance diagram for PiecewiseConstantParameter:



### 7.651.1 Detailed Description

Piecewise-constant parameter.

$a(t) = a_i$  if  $t_{i-1} \leq t < t_i$ . This kind of parameter is usually used to enhance the fitting of a model

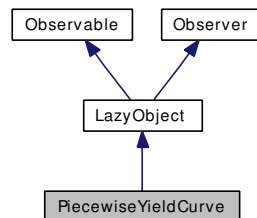
### Public Member Functions

- `PiecewiseConstantParameter` (const std::vector< [Time](#) > &times)

## 7.652 PiecewiseYieldCurve Class Template Reference

```
#include <ql/TermStructures/piecewiseyieldcurve.hpp>
```

Inheritance diagram for PiecewiseYieldCurve:



### 7.652.1 Detailed Description

```
template<class Traits, class Interpolator> class QuantLib::PiecewiseYieldCurve< Traits, Interpolator >
```

Piecewise yield term structure.

This term structure is bootstrapped on a number of interest rate instruments which are passed as a vector of handles to [RateHelper](#) instances. Their maturities mark the boundaries of the interpolated segments.

Each segment is determined sequentially starting from the earliest period to the latest and is chosen so that the instrument whose maturity marks the end of such segment is correctly repriced on the curve.

#### Warning

The bootstrapping algorithm will raise an exception if any two instruments have the same maturity date.

#### Tests

- the correctness of the returned values is tested by checking them against the original inputs.
- the observability of the term structure is tested.

## Public Member Functions

### YieldTermStructure interface

- `const std::vector< Date > & dates () const`
- `Date maxDate () const`
- `const std::vector< Time > & times () const`
- `Time maxTime () const`

### Observer interface

- `void update ()`

## Friends

- class `ObjectiveFunction`

## 7.652.2 Member Function Documentation

### 7.652.2.1 `void update ()` [virtual]

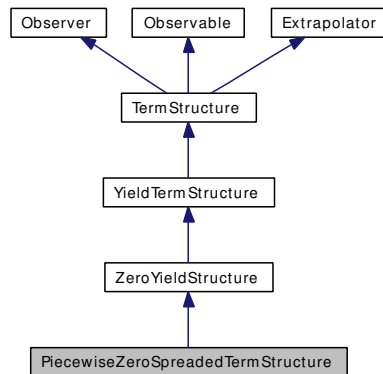
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [LazyObject](#).

## 7.653 PiecewiseZeroSpreadedTermStructure Class Reference

```
#include <ql/TermStructures/piecewisezerospreadedtermstructure.hpp>
```

Inheritance diagram for PiecewiseZeroSpreadedTermStructure:



### 7.653.1 Detailed Description

Term structure with an added vector of spreads on the zero-yield rate.

The zero-yield spread at any given date is linearly interpolated between the input data.

#### 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

- **PiecewiseZeroSpreadedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const std::vector< [Handle](#)< [Quote](#) > &spreads, const std::vector< [Date](#) > &dates)

#### YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Calendar](#) [calendar](#) () const  
*the calendar used for reference date calculation*
- const [Date](#) & [referenceDate](#) () const  
*the date at which discount = 1.0 and/or variance = 0.0*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return values*

## Protected Member Functions

- [Rate zeroYieldImpl](#) (Time) const  
*returns the spreaded zero yield rate*
- void [update](#) ()

## 7.653.2 Member Function Documentation

### 7.653.2.1 void update () [protected, virtual]

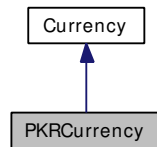
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

## 7.654 PKRCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for PKRCurrency:



### 7.654.1 Detailed Description

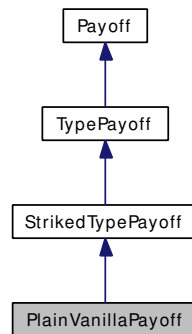
Pakistani rupee.

The ISO three-letter code is PKR; the numeric code is 586. It is divided in 100 paisa.

## 7.655 PlainVanillaPayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for PlainVanillaPayoff:



### 7.655.1 Detailed Description

Plain-vanilla payoff.

**Examples:**

[DiscreteHedging.cpp](#), [EquityOption.cpp](#), and [Replication.cpp](#).

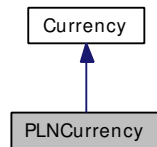
### Public Member Functions

- `PlainVanillaPayoff` (`Option::Type` type, `Real` strike)
- `Real operator()` (`Real` price) `const`
- virtual void `accept` ([AcyclicVisitor](#) &)

## 7.656 PLNCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for PLNCurrency:



### 7.656.1 Detailed Description

Polish zloty.

The ISO three-letter code is PLN; the numeric code is 985. It is divided in 100 groszy.



## 7.657 PoissonDistribution Class Reference

```
#include <ql/Math/poissondistribution.hpp>
```

### 7.657.1 Detailed Description

Normal distribution function.

Given an integer  $k$ , it returns its probability in a Poisson distribution.

#### Tests

the correctness of the returned value is tested by checking it against known good results.

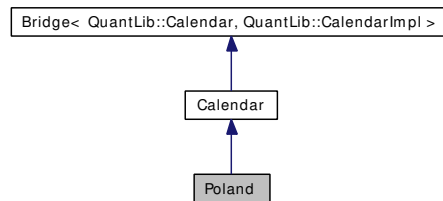
### Public Member Functions

- **PoissonDistribution** (Real mu)
- Real **operator()** (BigNatural k) const

## 7.658 Poland Class Reference

```
#include <ql/Calendars/poland.hpp>
```

Inheritance diagram for Poland:



### 7.658.1 Detailed Description

Polish calendar.

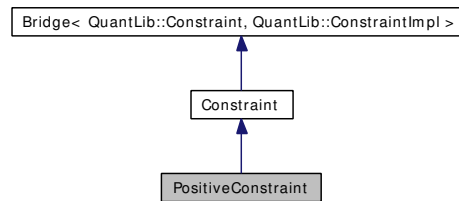
Holidays:

- 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

## 7.659 PositiveConstraint Class Reference

```
#include <ql/Optimization/constraint.hpp>
```

Inheritance diagram for PositiveConstraint:



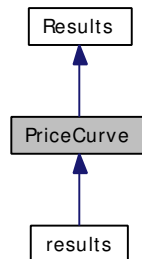
### 7.659.1 Detailed Description

Constraint imposing positivity to all arguments

## 7.660 PriceCurve Class Reference

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

Inheritance diagram for PriceCurve:



### 7.660.1 Detailed Description

additional pricing results

#### Public Member Functions

- void `reset()`

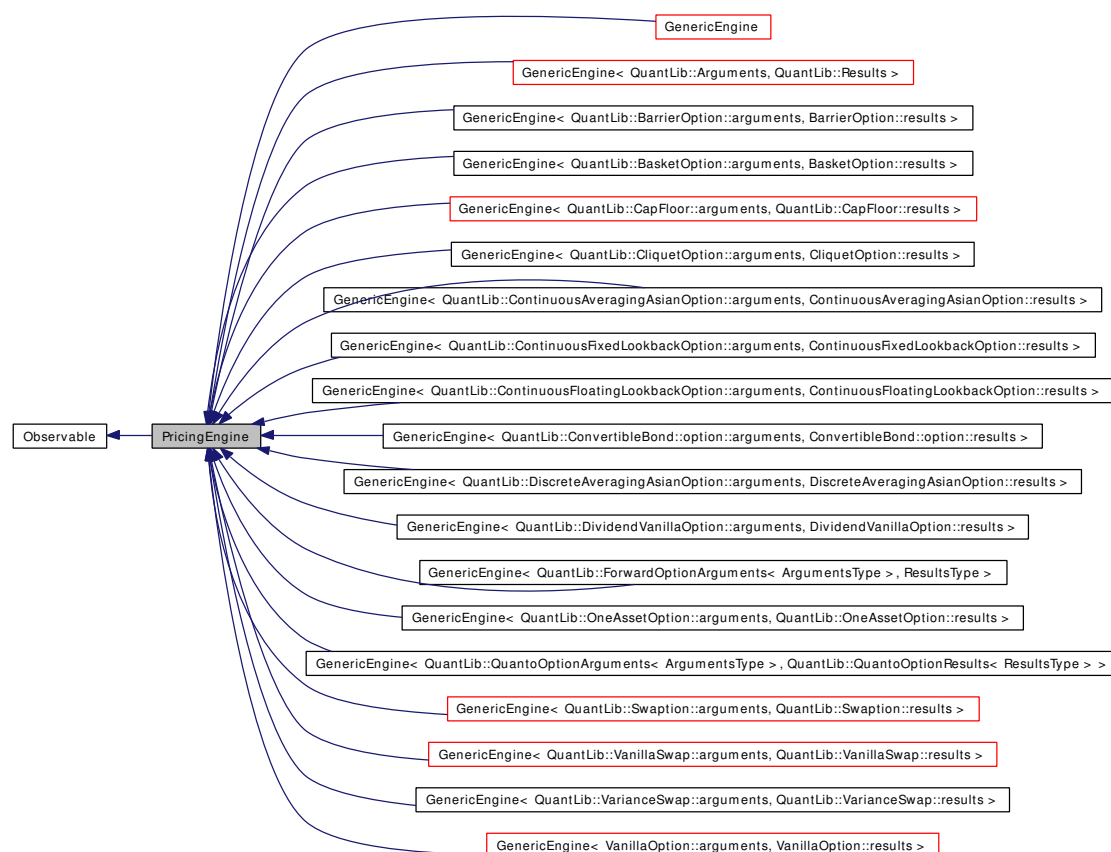
#### Public Attributes

- [SampledCurve](#) `priceCurve`

## 7.661 PricingEngine Class Reference

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

Inheritance diagram for PricingEngine:



### 7.661.1 Detailed Description

interface 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

## 7.662 PrimeNumbers Class Reference

```
#include <ql/Math/primenumbers.hpp>
```

### 7.662.1 Detailed Description

Prime numbers calculator.

Taken from "Monte Carlo Methods in Finance", by Peter Jäckel

### Static Public Member Functions

- static `BigNatural` [get](#) (Size `absoluteIndex`)  
*Get and store one after another.*

## 7.663 Problem Class Reference

```
#include <ql/Optimization/problem.hpp>
```

### 7.663.1 Detailed Description

Constrained optimization problem.

#### Public Member Functions

- [Problem](#) ([CostFunction](#) &f, [Constraint](#) &c, [OptimizationMethod](#) &meth)  
*default constructor*
- [Real value](#) (const [Array](#) &x) const  
*call cost function computation and increment evaluation counter*
- [Disposable](#)< [Array](#) > [values](#) (const [Array](#) &x) const  
*call cost values computation and increment evaluation counter*
- void [gradient](#) ([Array](#) &grad\_f, const [Array](#) &x) const  
*call cost function gradient computation and increment*
- [Real valueAndGradient](#) ([Array](#) &grad\_f, const [Array](#) &x) const  
*call cost function computation and it gradient*
- [OptimizationMethod](#) & [method](#) () const  
*Constrained optimization method.*
- [Constraint](#) & [constraint](#) () const  
*Constraint.*
- [CostFunction](#) & [costFunction](#) () const  
*Cost function.*
- void [minimize](#) () const  
*Minimization.*
- [Array](#) & [minimumValue](#) () const

#### Protected Attributes

- [CostFunction](#) & [costFunction\\_](#)  
*Unconstrained cost function.*
- [Constraint](#) & [constraint\\_](#)  
*Constraint.*
- [OptimizationMethod](#) & [method\\_](#)

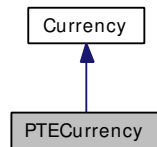
*constrained optimization method*



## 7.664 PTECurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for PTECurrency:



### 7.664.1 Detailed Description

Portuguese escudo.

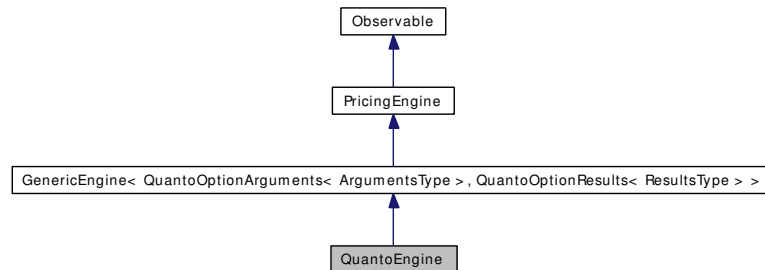
The ISO three-letter code was PTE; the numeric code was 620. It was divided in 100 centavos.

Obsoleted by the Euro since 1999.

## 7.665 QuantoEngine Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

Inheritance diagram for QuantoEngine:



### 7.665.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::QuantoEngine<
ArgumentsType, ResultsType >
```

Quanto engine base class.

#### Warning

for the time being, this engine will only work with simple Black-Scholes processes (i.e., no Merton.)

#### Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

### Public Member Functions

- **QuantoEngine** (const boost::shared\_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > &)
- void **calculate** () const
- ArgumentsType \* **underlyingArgs** () const

### Protected Attributes

- boost::shared\_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > > **originalEngine\_**
- ArgumentsType \* **originalArguments\_**
- const ResultsType \* **originalResults\_**

## 7.665.2 Member Function Documentation

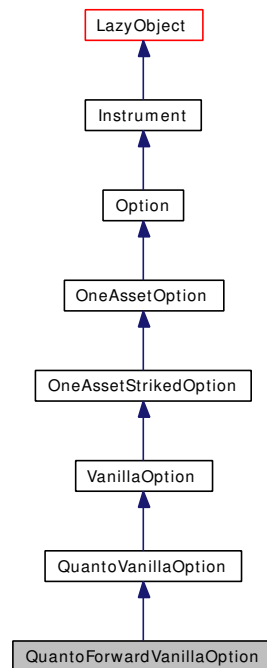
### 7.665.2.1 ArgumentsType\* underlyingArgs () const

Access to the arguments of the underlying engine is needed as this engine is not able to set them completely. When necessary, it must be done by the instrument: see [QuantoForwardVanillaOption](#) for an example.

## 7.666 QuantoForwardVanillaOption Class Reference

```
#include <ql/Instruments/quantoforwardvanillaoption.hpp>
```

Inheritance diagram for QuantoForwardVanillaOption:



### 7.666.1 Detailed Description

Quanto version of a forward vanilla option.

#### Public Types

- typedef [QuantoOptionArguments](#)< [ForwardVanillaOption::arguments](#) > **arguments**
- typedef [QuantoOptionResults](#)< [ForwardVanillaOption::results](#) > **results**
- typedef [QuantoEngine](#)< [ForwardVanillaOption::arguments](#), [ForwardVanillaOption::results](#) > **engine**

#### Public Member Functions

- **QuantoForwardVanillaOption** (const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &exchRateVolTS, const [Handle](#)< [Quote](#) > &correlation, Real moneyness, [Date](#) resetDate, const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &, const boost::shared\_ptr< [Exercise](#) > &, const boost::shared\_ptr< [PricingEngine](#) > &engine)
- void [setupArguments](#) ([Arguments](#) \*) const

## 7.666.2 Member Function Documentation

### 7.666.2.1 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](#).

## 7.667 QuantoOptionArguments Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

### 7.667.1 Detailed Description

```
template<class ArgumentsType> class QuantLib::QuantoOptionArguments< ArgumentsType  
>
```

Arguments for quanto option calculation

### Public Member Functions

- void `validate ()` const

### Public Attributes

- Real `correlation`
- [Handle](#)< [YieldTermStructure](#) > `foreignRiskFreeTS`
- [Handle](#)< [BlackVolTermStructure](#) > `exchRateVolTS`

## 7.668 QuantoOptionResults Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

### 7.668.1 Detailed Description

```
template<class ResultsType> class QuantLib::QuantoOptionResults< ResultsType >
```

Results from quanto option calculation

#### Public Member Functions

- void reset ()

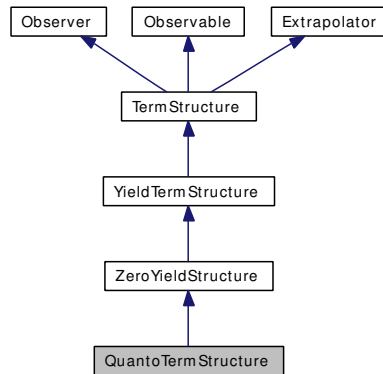
#### Public Attributes

- Real qvega
- Real qrho
- Real qlambda

## 7.669 QuantoTermStructure Class Reference

```
#include <ql/TermStructures/quantotermstructure.hpp>
```

Inheritance diagram for QuantoTermStructure:



### 7.669.1 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 [Handle](#)< [YieldTermStructure](#) > &underlyingDividendTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &underlyingBlackVolTS, Real strike, const [Handle](#)< [BlackVolTermStructure](#) > &exchRateBlackVolTS, Real exchRate-ATMlevel, Real underlyingExchRateCorrelation)

#### YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Calendar](#) [calendar](#) () const  
*the calendar used for reference date calculation*
- const [Date](#) & [referenceDate](#) () const  
*the date at which discount = 1.0 and/or variance = 0.0*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return values*



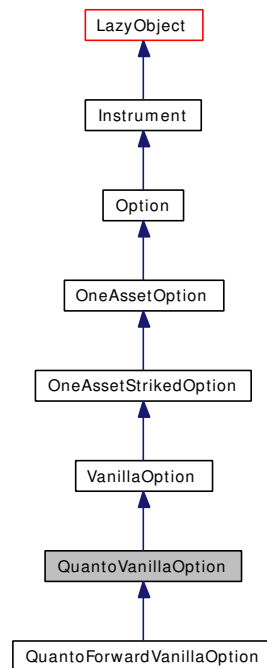
## Protected Member Functions

- [Rate zeroYieldImpl](#) (Time) const  
*returns the zero yield as seen from the evaluation date*

## 7.670 QuantoVanillaOption Class Reference

```
#include <ql/Instruments/quantovanillaoption.hpp>
```

Inheritance diagram for QuantoVanillaOption:



### 7.670.1 Detailed Description

quanto version of a vanilla option

#### Public Types

- typedef [QuantoOptionArguments](#)< [VanillaOption::arguments](#) > **arguments**
- typedef [QuantoOptionResults](#)< [VanillaOption::results](#) > **results**
- typedef [QuantoEngine](#)< [VanillaOption::arguments](#), [VanillaOption::results](#) > **engine**

#### Public Member Functions

- **QuantoVanillaOption** (const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &exchRateVolTS, const [Handle](#)< [Quote](#) > &correlation, const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &, const boost::shared\_ptr< [Exercise](#) > &, const boost::shared\_ptr< [PricingEngine](#) > &)
- void [setupArguments](#) ([Arguments](#) \*) const
- void [fetchResults](#) (const [Results](#) \*) const

**greeks**

- Real **qvega** () const
- Real **qrho** () const
- Real **qlambda** () const

## Protected Member Functions

- void **setupExpired** () const

## Protected Attributes

- [Handle< YieldTermStructure >](#) **foreignRiskFreeTS\_**
- [Handle< BlackVolTermStructure >](#) **exchRateVolTS\_**
- [Handle< Quote >](#) **correlation\_**
- Real **qvega\_**
- Real **qrho\_**
- Real **qlambda\_**

## 7.670.2 Member Function Documentation

### 7.670.2.1 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 [OneAssetStrikedOption](#).

Reimplemented in [QuantoForwardVanillaOption](#).

### 7.670.2.2 void **fetchResults** (const [Results](#) \*) const [virtual]

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

Reimplemented from [OneAssetStrikedOption](#).

### 7.670.2.3 void **setupExpired** () const [protected, virtual]

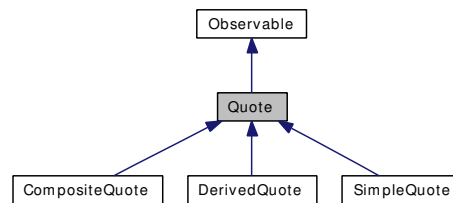
This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [OneAssetStrikedOption](#).

## 7.671 Quote Class Reference

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

Inheritance diagram for Quote:



### 7.671.1 Detailed Description

purely virtual base class for market observables

#### Tests

the observability of class instances is tested.

### Public Member Functions

- virtual [Real value](#) () const=0  
*returns the current value*

## 7.672 RandomizedLDS Class Template Reference

```
#include <ql/RandomNumbers/randomizedlds.hpp>
```

### 7.672.1 Detailed Description

```
template<class LDS, class PRS = RandomSequenceGenerator<MersenneTwisterUniform-
Rng>> class QuantLib::RandomizedLDS< LDS, PRS >
```

Randomized (random shift) low-discrepancy sequence.

Random-shifts a uniform low-discrepancy sequence of dimension  $N$  by adding (modulo 1 for each coordinate) a pseudo-random uniform deviate in  $(0, 1)^N$ . It is used for implementing Randomized Quasi Monte Carlo.

The uniform low discrepancy sequence is supplied by LDS; the uniform pseudo-random sequence is supplied by PRS.

Both class LDS and PRS must implement the following interface:

```
LDS::sample_type LDS::nextSequence() const;
Size LDS::dimension() const;
```

#### Precondition:

LDS and PRS must have the same dimension  $N$

#### Warning

Inverting LDS and PRS is possible, but it doesn't make sense.

#### Todo

implement the other randomization algorithms

#### Tests

correct initialization is tested.

### Public Types

- typedef [Sample< Array >](#) **sample\_type**

### Public Member Functions

- **RandomizedLDS** (const LDS &lds, const PRS &prsg)
- **RandomizedLDS** (const LDS &lds)
- **RandomizedLDS** (Size dimensionality, BigNatural ldsSeed=0, BigNatural prsSeed=0)
- const [sample\\_type](#) & **nextSequence** () const  
*returns next sample using a given randomizing vector*
- const [sample\\_type](#) & **lastSequence** () const
- void **nextRandomizer** ()
- Size **dimension** () const

## 7.672.2 Member Function Documentation

### 7.672.2.1 void nextRandomizer ()

update the randomizing vector and re-initialize the low discrepancy generator

## 7.673 RandomSequenceGenerator Class Template Reference

```
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
```

### 7.673.1 Detailed Description

```
template<class RNG> class QuantLib::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 [Sample< Array >](#) **sample\_type**

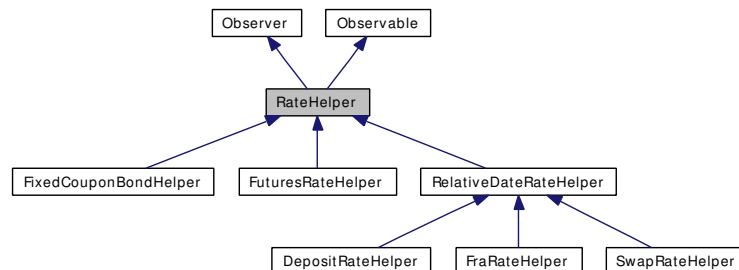
### Public Member Functions

- **RandomSequenceGenerator** (Size dimensionality, const RNG &rng)
- **RandomSequenceGenerator** (Size dimensionality, BigNatural seed=0)
- const [sample\\_type](#) & **nextSequence** () const
- std::vector< BigNatural > **nextInt32Sequence** () const
- const [sample\\_type](#) & **lastSequence** () const
- Size **dimension** () const

## 7.674 RateHelper Class Reference

```
#include <ql/TermStructures/piecewiseyieldcurve.hpp>
```

Inheritance diagram for RateHelper:



### 7.674.1 Detailed Description

Base helper class for yield-curve bootstrapping.

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 consistency between the algorithms used during bootstrapping and later instrument pricing. This is not yet fully enforced in the available rate helpers, though - only [SwapRateHelper](#) and [FixedCouponBondHelper](#) contain their corresponding instrument for the time being.

### Public Member Functions

- [RateHelper](#) (const [Handle](#)< [Quote](#) > &quote)
- [RateHelper](#) ([Real](#) quote)

#### RateHelper interface

- [Real](#) [quoteError](#) () const
- [Real](#) [referenceQuote](#) () const
- virtual [Real](#) [impliedQuote](#) () const=0
- virtual [DiscountFactor](#) [discountGuess](#) () const
- virtual void [setTermStructure](#) ([YieldTermStructure](#) \*)  
*sets the term structure to be used for pricing*
- virtual [Date](#) [earliestDate](#) () const  
*earliest relevant date*
- virtual [Date](#) [latestDate](#) () const  
*latest relevant date*

#### Observer interface

- virtual void [update](#) ()



## Protected Attributes

- [Handle< Quote >](#) quote\_
- [YieldTermStructure](#) \* termStructure\_
- [Date](#) earliestDate\_
- [Date](#) latestDate\_

## 7.674.2 Member Function Documentation

### 7.674.2.1 virtual void setTermStructure ([YieldTermStructure](#) \*) [virtual]

sets the term structure to be used for pricing

#### Warning

Being a pointer and not a shared\_ptr, 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 [FixedCouponBondHelper](#), and [SwapRateHelper](#).

### 7.674.2.2 virtual [Date](#) earliestDate () const [virtual]

earliest relevant date

The earliest date at which discounts are needed by the helper in order to provide a quote.

### 7.674.2.3 virtual [Date](#) latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Reimplemented in [FixedCouponBondHelper](#).

### 7.674.2.4 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.

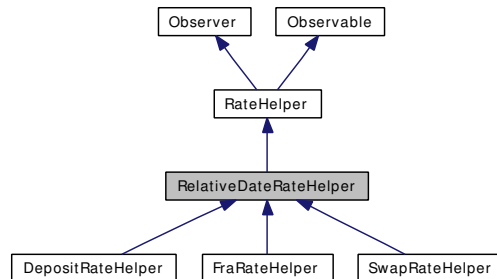
Implements [Observer](#).

Reimplemented in [RelativeDateRateHelper](#).

## 7.675 RelativeDateRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for RelativeDateRateHelper:



### 7.675.1 Detailed Description

Rate helper with date schedule relative to the global evaluation date.

This class takes care of rebuilding the date schedule when the global evaluation date changes

#### Public Member Functions

- **RelativeDateRateHelper** (const [Handle](#)< [Quote](#) > &quote)
- **RelativeDateRateHelper** ([Real](#) quote)
- void [update](#) ()

#### Protected Member Functions

- virtual void **initializeDates** ()=0

#### Protected Attributes

- [Date](#) **evaluationDate\_**

### 7.675.2 Member Function Documentation

#### 7.675.2.1 void update () [virtual]

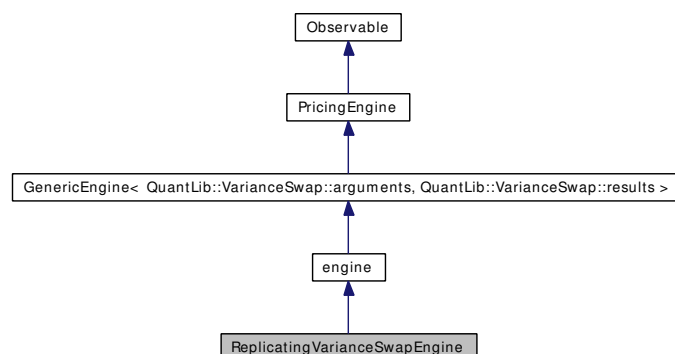
This method must be implemented in derived classes. An instance of [Observer](#) does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [RateHelper](#).

## 7.676 ReplicatingVarianceSwapEngine Class Reference

```
#include <ql/PricingEngines/Forward/replicatingvarianceswapengine.hpp>
```

Inheritance diagram for ReplicatingVarianceSwapEngine:



### 7.676.1 Detailed Description

Variance-swap pricing engine using replicating cost,.

as described in Demeterfi, Derman, Kamal & Zou, "A Guide to Volatility and Variance Swaps", 1999

#### Tests

returned fair variances verified against results from literature

### Public Types

- typedef std::vector< Real > **StrikesType**

### Public Member Functions

- **ReplicatingVarianceSwapEngine** (const Real dk=5.0, const StrikesType &callStrikes=StrikesType(), const StrikesType &putStrikes=StrikesType(), const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())
- void **calculate** () const

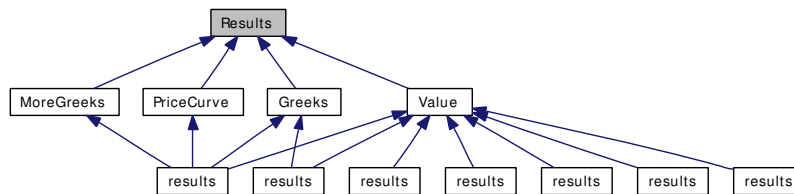
### Protected Member Functions

- void **computeOptionWeights** (const StrikesType &, const Option::Type) const
- Real **computeLogPayoff** (const Real, const Real) const
- Real **computeReplicatingPortfolio** () const
- [Rate](#) **riskFreeRate** () const
- [DiscountFactor](#) **riskFreeDiscount** () const
- Real **underlying** () const
- [Time](#) **residualTime** () const

## 7.677 Results Class Reference

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

Inheritance diagram for Results:



### 7.677.1 Detailed Description

base class for generic result groups

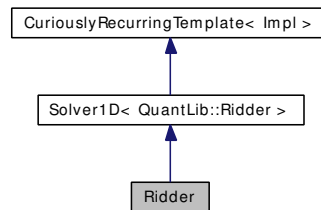
#### Public Member Functions

- virtual void **reset** ()=0

## 7.678 Ridder Class Reference

```
#include <ql/Solvers1D/ridder.hpp>
```

Inheritance diagram for Ridder:



### 7.678.1 Detailed Description

Ridder 1-D solver

#### Tests

the correctness of the returned values is tested by checking them against known good results.

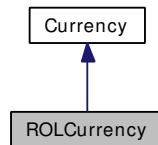
### Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAcc) const`

## 7.679 ROLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ROLCurrency:



### 7.679.1 Detailed Description

Romanian leu.

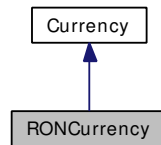
The ISO three-letter code was ROL; the numeric code was 642. It was divided in 100 bani.

Obsoleted by the new leu since July 2005.

## 7.680 RONCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for RONCurrency:



### 7.680.1 Detailed Description

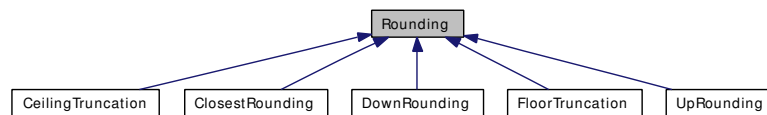
Romanian new leu.

The ISO three-letter code is RON; the numeric code is 946. It is divided in 100 bani.

## 7.681 Rounding Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for Rounding:



### 7.681.1 Detailed Description

basic rounding class

#### Tests

the correctness of the returned values is tested by checking them against known good results.

#### Inspectors

- Integer **precision** () const
- **Type** **type** () const
- Integer **roundingDigit** () const

#### Public Types

- enum **Type** {  
     None, Up, Down, Closest,  
     Floor, Ceiling }  
*rounding methods*

#### Public Member Functions

- **Rounding** ()  
*default constructor*
- **Rounding** (Integer precision, **Type** type=Closest, Integer digit=5)
- Decimal **operator()** (Decimal value) const  
*perform rounding*

### 7.681.2 Member Enumeration Documentation

#### 7.681.2.1 enum **Type**

rounding methods



The rounding methods follow the OMG specification available at <ftp://ftp.omg.org/pub/docs/formal/00-06-29.pdf>

### Warning

the names of the [Floor](#) and Ceiling methods might be misleading. Check the provided reference.

### Enumerator:

*None* do not round: return the number unmodified

*Up* the first decimal place past the precision will be rounded up. This differs from the OMG rule which rounds up only if the decimal to be rounded is greater than or equal to the rounding digit

*Down* all decimal places past the precision will be truncated

*Closest* the first decimal place past the precision will be rounded up if greater than or equal to the rounding digit; this corresponds to the OMG round-up rule. When the rounding digit is 5, the result will be the one closest to the original number, hence the name.

*Floor* positive numbers will be rounded up and negative numbers will be rounded down using the OMG round up and round down rules

*Ceiling* positive numbers will be rounded down and negative numbers will be rounded up using the OMG round up and round down rules

## 7.681.3 Constructor & Destructor Documentation

### 7.681.3.1 [Rounding \(\)](#)

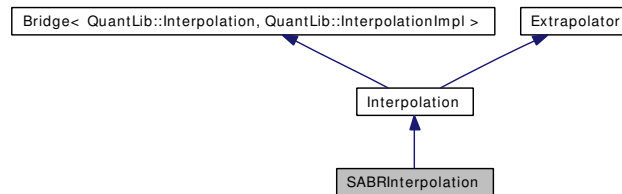
default constructor

Instances built through this constructor don't perform any rounding.

## 7.682 SABRInterpolation Class Reference

```
#include <ql/Math/sabrinterpolation.hpp>
```

Inheritance diagram for SABRInterpolation:



### 7.682.1 Detailed Description

SABR smile interpolation between discrete volatility points.

#### Public Member Functions

- `template<class I1, class I2> SABRInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, Time t, Real forward, Real alpha, Real beta, Real nu, Real rho, bool isAlphaFixed, bool isBetaFixed, bool isNuFixed, bool isRhoFixed, const boost::shared_ptr< OptimizationMethod > &method=boost::shared_ptr< OptimizationMethod >())`
- `Real expiry () const`
- `Real forward () const`
- `Real alpha () const`
- `Real beta () const`
- `Real nu () const`
- `Real rho () const`
- `Real interpolationError () const`
- `Real interpolationMaxError () const`
- `EndCriteria::Type endCriteria ()`

## 7.683 SalvagingAlgorithm Struct Reference

```
#include <ql/Math/pseudosqrt.hpp>
```

### 7.683.1 Detailed Description

algorithm used for matricial pseudo square root

#### Public Types

- enum Type { None, Spectral, Hypersphere, LowerDiagonal }

## 7.684 Sample Struct Template Reference

```
#include <ql/MonteCarlo/sample.hpp>
```

### 7.684.1 Detailed Description

```
template<class T> struct QuantLib::Sample< T >
```

weighted sample

#### Public Types

- typedef T **value\_type**

#### Public Member Functions

- **Sample** (const T &value, Real weight)

#### Public Attributes

- T **value**
- Real **weight**

## 7.685 SampledCurve Class Reference

```
#include <ql/Math/sampledcurve.hpp>
```

### 7.685.1 Detailed Description

This class contains a sampled curve.

Initially the class will contain one indexed curve

### Public Member Functions

- **SampledCurve** (Size gridSize=0)
- **SampledCurve** (const [Array](#) &grid)
- **SampledCurve** & **operator=** (const [SampledCurve](#) &)

#### inspectors

- const [Array](#) & **grid** () const
- [Array](#) & **grid** ()
- const [Array](#) & **values** () const
- [Array](#) & **values** ()
- [Real](#) **gridValue** (Size i) const
- [Real](#) & **gridValue** (Size i)
- [Real](#) **value** (Size i) const
- [Real](#) & **value** (Size i)
- Size **size** () const
- bool **empty** () const

#### modifiers

- void **setGrid** (const [Array](#) &)
- void **setValues** (const [Array](#) &)
- template<class F> void **sample** (const F &f)

#### calculations

- [Real](#) **valueAtCenter** () const
- [Real](#) **firstDerivativeAtCenter** () const
- [Real](#) **secondDerivativeAtCenter** () const

#### utilities

- void **swap** ([SampledCurve](#) &)
- void **setLogGrid** ([Real](#) min, [Real](#) max)
- void **regridLogGrid** ([Real](#) min, [Real](#) max)
- void **shiftGrid** ([Real](#) s)
- void **scaleGrid** ([Real](#) s)
- void **regrid** (const [Array](#) &new\_grid)
- template<class T> void **regrid** (const [Array](#) &new\_grid, T func)
- template<class T> const [SampledCurve](#) & **transform** (T x)
- template<class T> const [SampledCurve](#) & **transformGrid** (T x)

## 7.685.2 Member Function Documentation

### 7.685.2.1 **Real** valueAtCenter () const

#### **Todo**

replace or complement with a more general function valueAt(spot)

### 7.685.2.2 **Real** firstDerivativeAtCenter () const

#### **Todo**

replace or complement with a more general function firstDerivativeAt(spot)

### 7.685.2.3 **Real** secondDerivativeAtCenter () const

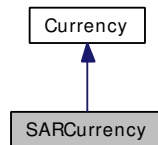
#### **Todo**

replace or complement with a more general function secondDerivativeAt(spot)

## 7.686 SARCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for SARCurrency:



### 7.686.1 Detailed Description

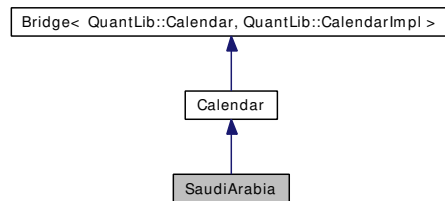
Saudi riyal.

The ISO three-letter code is SAR; the numeric code is 682. It is divided in 100 halalat.

## 7.687 SaudiArabia Class Reference

```
#include <ql/Calendars/saudiarabia.hpp>
```

Inheritance diagram for SaudiArabia:



### 7.687.1 Detailed Description

Saudi Arabian calendar.

Holidays:

- Fridays

Other holidays for which no rule is given (data available for 2004-2005 only:)

- EID AL-ADHA
- EID AL-FITR



## 7.688 Schedule Class Reference

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

### 7.688.1 Detailed Description

Payment schedule.

**Examples:**

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

### 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, [Frequency](#) frequency, [BusinessDayConvention](#) convention, const [Date](#) &stubDate=[Date](#)(), bool startFromEnd=false, bool longFinal=false)
- [Schedule](#) (const [Calendar](#) &calendar, const [Date](#) &startDate, const [Date](#) &endDate, const [Period](#) &tenor, [BusinessDayConvention](#) convention, const [Date](#) &stubDate=[Date](#)(), bool startFromEnd=false, bool longFinal=false)
- [Frequency](#) **frequency** () const
- [Schedule](#) (const std::vector< [Date](#) > &, const [Calendar](#) &calendar=[NullCalendar](#)(), [BusinessDayConvention](#) convention=[Unadjusted](#))
- [Schedule](#) (const [Date](#) &effectiveDate, const [Date](#) &terminationDate, const [Period](#) &tenor, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, [BusinessDayConvention](#) terminationDateConvention, bool backward, bool endOfMonth, const [Date](#) &firstDate=[Date](#)(), const [Date](#) &nextToLastDate=[Date](#)())

### Date access

- Size **size** () const
- const [Date](#) & **operator**[ ] (Size i) const
- const [Date](#) & **at** (Size i) const
- const [Date](#) & **date** (Size i) const
- const std::vector< [Date](#) > & **dates** () const
- bool **isRegular** (Size i) const

### Other inspectors

- const [Calendar](#) & **calendar** () const
- const [Date](#) & **startDate** () const
- const [Date](#) & **endDate** () const
- const [Period](#) & **tenor** () const
- [BusinessDayConvention](#) **businessDayConvention** () const

## 7.688.2 Constructor & Destructor Documentation

7.688.2.1 **Schedule** (const **Calendar** & *calendar*, const **Date** & *startDate*, const **Date** & *endDate*, **Frequency** *frequency*, **BusinessDayConvention** *convention*, const **Date** & *stubDate* = **Date**(), bool *startFromEnd* = false, bool *longFinal* = false)

### Deprecated

use other constructors instead

7.688.2.2 **Schedule** (const **Calendar** & *calendar*, const **Date** & *startDate*, const **Date** & *endDate*, const **Period** & *tenor*, **BusinessDayConvention** *convention*, const **Date** & *stubDate* = **Date**(), bool *startFromEnd* = false, bool *longFinal* = false)

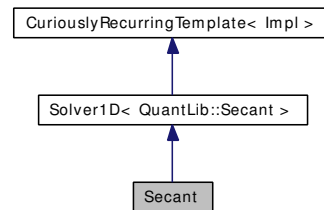
### Deprecated

use other constructors instead

## 7.689 Secant Class Reference

```
#include <ql/Solvers1D/secant.hpp>
```

Inheritance diagram for Secant:



### 7.689.1 Detailed Description

Secant 1-D solver

#### Tests

the correctness of the returned values is tested by checking them against known good results.

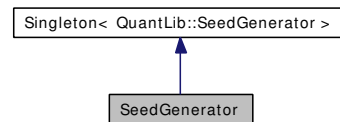
### Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

## 7.690 SeedGenerator Class Reference

```
#include <ql/RandomNumbers/seedgenerator.hpp>
```

Inheritance diagram for SeedGenerator:



### 7.690.1 Detailed Description

Random seed generator.

Random number generator used for automatic generation of initialization seeds.

#### Tests

correct initializaion of the single instance is tested.

### Public Member Functions

- unsigned long `get()`

### Friends

- class `Singleton< SeedGenerator >`

## 7.691 SegmentIntegral Class Reference

```
#include <ql/Math/segmentintegral.hpp>
```

### 7.691.1 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

$$\int_a^b f dx = \frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \dots + f(x_{N-1}) + \frac{1}{2}f(x_N)$$

where  $x_0 = a$ ,  $x_N = b$ , and  $x_i = a + i\Delta x$  with  $\Delta x = (b - a)/N$ .

#### Tests

the correctness of the result is tested by checking it against known good values.

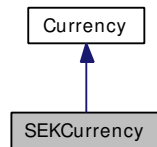
### Public Member Functions

- **SegmentIntegral** (Size intervals)
- `template<class F> Real operator() (const F &f, Real a, Real b) const`

## 7.692 SEKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for SEKCurrency:



### 7.692.1 Detailed Description

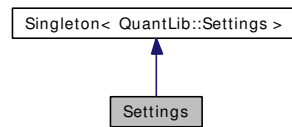
Swedish krona.

The ISO three-letter code is SEK; the numeric code is 752. It is divided in 100 öre.

## 7.693 Settings Class Reference

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

Inheritance diagram for Settings:



### 7.693.1 Detailed Description

global repository for run-time library settings

#### Public Member Functions

- DateProxy & [evaluationDate](#) ()  
*the date at which pricing is to be performed.*
- const DateProxy & [evaluationDate](#) () const

#### Friends

- class [Singleton](#)< [Settings](#) >
- std::ostream & [operator](#)<< (std::ostream &, const DateProxy &)

### 7.693.2 Member Function Documentation

#### 7.693.2.1 Settings::DateProxy & evaluationDate ()

the date at which pricing is to be performed.

Client code can inspect the evaluation date, as in:

```
Date d = Settings::instance().evaluationDate();
```

where today's date is returned if the evaluation date is set to the null date (its default value;) can set it to a new value, as in:

```
Settings::instance().evaluationDate() = d;
```

and can register with it, as in:

```
registerWith(Settings::instance().evaluationDate());
```

to be notified when it is set to a new value.

**Warning**

a notification is not sent when the evaluation date changes for natural causes—i.e., a date was not explicitly set (which results in today's date being used for pricing) and the current date changes as the clock strikes midnight.



## 7.694 Settlement Struct Reference

```
#include <ql/Instruments/swaption.hpp>
```

### 7.694.1 Detailed Description

settlement information

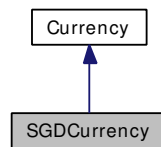
#### Public Types

- enum Type { Physical, Cash }

## 7.695 SGDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for SGDCurrency:



### 7.695.1 Detailed Description

[Singapore](#) dollar.

The ISO three-letter code is SGD; the numeric code is 702. It is divided in 100 cents.

## 7.696 Short Class Template Reference

```
#include <ql/CashFlows/shortindexedcoupon.hpp>
```

### 7.696.1 Detailed Description

**template<class IndexedCouponType> class QuantLib::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

- **template<class IndexType> Short** (const [Date](#) &paymentDate, const [Real](#) nominal, const [Date](#) &startDate, const [Date](#) &endDate, const [Integer](#) fixingDays, const boost::shared\_ptr< IndexType > &index, const [Real](#) gearing=1.0, const [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())
- **[Real](#) amount** () const  
*inhibit calculation*

### 7.696.2 Member Function Documentation

#### 7.696.2.1 [Real](#) amount () const

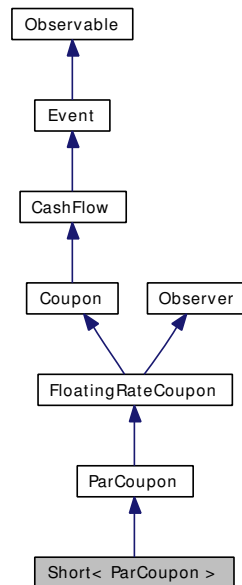
*inhibit calculation*

Unlike [ParCoupon](#), this coupon can't calculate its fixing for future dates, either.

## 7.697 Short< ParCoupon > Class Template Reference

```
#include <ql/CashFlows/shortfloatingcoupon.hpp>
```

Inheritance diagram for Short< ParCoupon >:



### 7.697.1 Detailed Description

```
template<> class QuantLib::Short< ParCoupon >
```

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

- **Short** (const [Date](#) &paymentDate, const [Real](#) nominal, const [Date](#) &startDate, const [Date](#) &endDate, const [Integer](#) fixingDays, const boost::shared\_ptr< [Xibor](#) > &index, const [Real](#) gearing=1.0, const [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())
- **Real rate** () const

*throws when an interpolated fixing is needed*

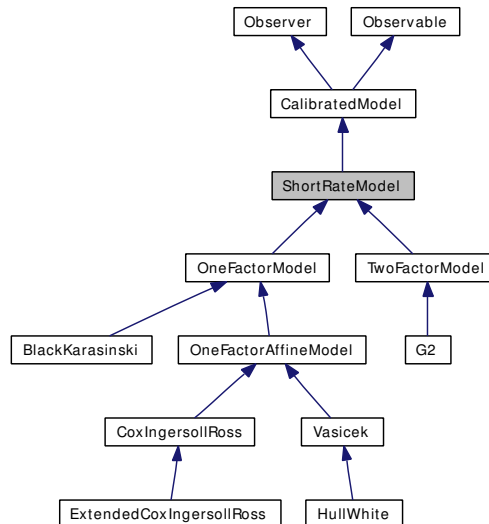
### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.698 ShortRateModel Class Reference

```
#include <ql/ShortRateModels/model.hpp>
```

Inheritance diagram for ShortRateModel:



### 7.698.1 Detailed Description

Abstract short-rate model class.

#### Public Member Functions

- **ShortRateModel** (Size nArguments)
- virtual boost::shared\_ptr< [NumericalMethod](#) > **tree** (const [TimeGrid](#) &) const=0

## 7.699 ShoutCondition Class Reference

```
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

### 7.699.1 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.

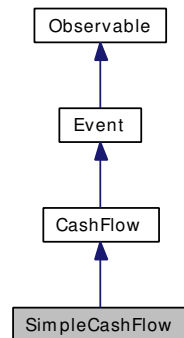
### Public Member Functions

- **ShoutCondition** (Option::Type type, Real strike, Time resTime, Rate rate)
- **ShoutCondition** (const [Array](#) &intrinsicValues, Time resTime, Rate rate)
- void **applyTo** ([Array](#) &a, Time t) const

## 7.700 SimpleCashFlow Class Reference

```
#include <ql/CashFlows/simplecashflow.hpp>
```

Inheritance diagram for SimpleCashFlow:



### 7.700.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

### Public Member Functions

- **SimpleCashFlow** (Real amount, const [Date](#) &date)

#### CashFlow interface

- Real [amount](#) () const  
*returns the amount of the cash flow*
- [Date](#) [date](#) () const  
*Note:*  
*This is inherited from the event class*

#### Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

### 7.700.2 Member Function Documentation

#### 7.700.2.1 Real amount () const [virtual]

returns the amount of the cash flow

#### Note:

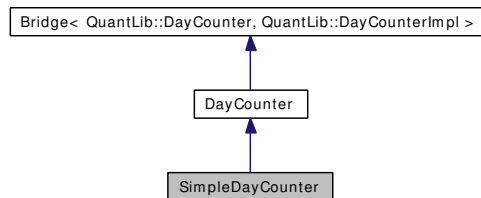
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

## 7.701 SimpleDayCounter Class Reference

```
#include <ql/DayCounters/simpliedaycounter.hpp>
```

Inheritance diagram for SimpleDayCounter:



### 7.701.1 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.

#### Tests

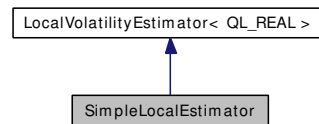
the correctness of the results is checked against known good values.



## 7.702 SimpleLocalEstimator Class Reference

```
#include <ql/VolatilityModels/simplelocalestimator.hpp>
```

Inheritance diagram for SimpleLocalEstimator:



### 7.702.1 Detailed Description

This class implements a concrete volatility model

Volatilities are assumed to be expressed on an annual basis.

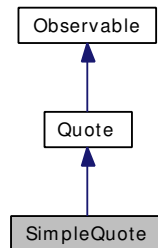
#### Public Member Functions

- **SimpleLocalEstimator** (Real y)
- `TimeSeries< Volatility > calculate` (const `TimeSeries< Real >` &quoteSeries)

## 7.703 SimpleQuote Class Reference

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

Inheritance diagram for SimpleQuote:



### 7.703.1 Detailed Description

market element returning a stored value

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), [EquityOption.cpp](#), [FRA.cpp](#), [Replication.cpp](#), and [swapvaluation.cpp](#).

### Public Member Functions

- **SimpleQuote** (Real value)

#### Quote interface

- Real **value** () const  
*returns the current value*

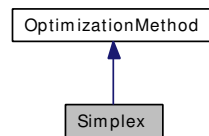
#### Modifiers

- Real **setValue** (Real value)  
*returns the difference between the new value and the old value*

## 7.704 Simplex Class Reference

```
#include <ql/Optimization/simplex.hpp>
```

Inheritance diagram for Simplex:



### 7.704.1 Detailed Description

Multi-dimensional simplex class.

#### Public Member Functions

- [Simplex](#) (Real lambda, Real tol)
- void [minimize](#) (const [Problem](#) &P) const  
*minimize the optimization problem P*

### 7.704.2 Constructor & Destructor Documentation

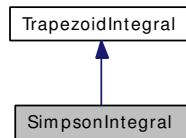
#### 7.704.2.1 [Simplex](#) (Real *lambda*, Real *tol*)

Constructor taking as input the characteristic length and tolerance

## 7.705 SimpsonIntegral Class Reference

```
#include <ql/Math/simpsonintegral.hpp>
```

Inheritance diagram for SimpsonIntegral:



### 7.705.1 Detailed Description

Integral of a one-dimensional function.

#### Tests

the correctness of the result is tested by checking it against known good values.

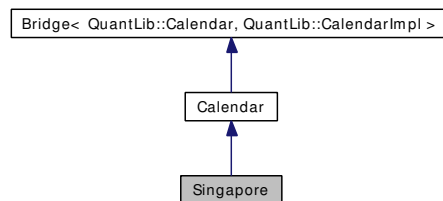
### Public Member Functions

- **SimpsonIntegral** (Real accuracy, Size maxIterations=[Null](#)< Size >())
- **operator()** (const F &f, Real a, Real b) const

## 7.706 Singapore Class Reference

```
#include <ql/Calendars/singapore.hpp>
```

Inheritance diagram for Singapore:



### 7.706.1 Detailed Description

Singapore calendars

Holidays for the [Singapore](http://www.ses.com.sg) exchange (data from [<http://www.ses.com.sg>](http://www.ses.com.sg)):

- Saturdays
- Sundays
- New Year's day, January 1st
- Good Friday
- Labour Day, May 1st
- National Day, August 9th
- Christmas, December 25th
- Boxing Day, December 26th

Other holidays for which no rule is given (data available for 2004-2005 only:)

- Chinese New Year
- Hari Raya Haji
- Vesak Poya Day
- Deepavali
- Diwali
- Hari Raya Puasa

### Public Types

- enum [Market](#) { [SGX](#) }

## Public Member Functions

- Singapore ([Market](#) m=SGX)

## 7.706.2 Member Enumeration Documentation

### 7.706.2.1 enum [Market](#)

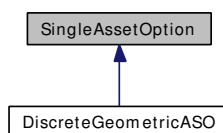
Enumerator:

SGX [Singapore](#) exchange.

## 7.707 SingleAssetOption Class Reference

#include <ql/Pricers/singleassetoption.hpp>

Inheritance diagram for SingleAssetOption:



### 7.707.1 Detailed Description

Black-Scholes-Merton option.

#### Public Member Functions

- **SingleAssetOption** (Option::Type type, Real underlying, Real strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, Volatility volatility)
- virtual void **setVolatility** (Volatility newVolatility)
- virtual void **setRiskFreeRate** (Rate newRate)
- virtual void **setDividendYield** (Rate newDividendYield)
- virtual Real **value** () const=0
- virtual Real **delta** () const=0
- virtual Real **gamma** () const=0
- virtual Real **theta** () const
- virtual Real **vega** () const
- virtual Real **rho** () const
- virtual Real **dividendRho** () const
- Volatility **impliedVolatility** (Real targetValue, Real accuracy=1e-4, Size maxEvaluations=100, Volatility minVol=QL\_MIN\_VOLATILITY, Volatility maxVol=QL\_MAX\_VOLATILITY) const
- Spread **impliedDivYield** (Real targetValue, Real accuracy=1e-4, Size maxEvaluations=100, Spread minYield=QL\_MIN\_DIVYIELD, Spread maxYield=QL\_MAX\_DIVYIELD) const
- virtual boost::shared\_ptr< [SingleAssetOption](#) > **clone** () const=0

#### Protected Attributes

- Real **underlying\_**
- [PlainVanillaPayoff](#) **payoff\_**
- Spread **dividendYield\_**
- Rate **riskFreeRate\_**
- Time **residualTime\_**
- Volatility **volatility\_**
- bool **hasBeenCalculated\_**
- Real **rho\_**
- Real **dividendRho\_**
- Real **vega\_**

- Real `theta_`
- bool `rhoComputed_`
- bool `dividendRhoComputed_`
- bool `vegaComputed_`
- bool `thetaComputed_`

### Static Protected Attributes

- static const Real `dVolMultiplier_`
- static const Real `dRMultiplier_`

### Friends

- class `VolatilityFunction`
- class `DivYieldFunction`

## 7.707.2 Member Function Documentation

**7.707.2.1** `Volatility` `impliedVolatility` (Real *targetValue*, Real *accuracy* = 1e-4, Size *maxEvaluations* = 100, Volatility *minVol* = QL\_MIN\_VOLATILITY, Volatility *maxVol* = QL\_MAX\_VOLATILITY) const

### Warning

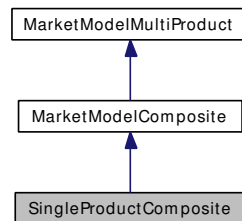
Options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g. binary options. In these cases `impliedVolatility` can fail and in any case is meaningless. Another possible source of failure is to have a `targetValue` that is not attainable with any volatility, e.g. a `targetValue` lower than the intrinsic value in the case of American options.



## 7.708 SingleProductComposite Class Reference

```
#include <ql/MarketModels/Products/singleproductcomposite.hpp>
```

Inheritance diagram for SingleProductComposite:



### 7.708.1 Detailed Description

Composition of one or more market-model products.

Instances of this class build a single market-model product by composing two or more subproducts.

#### Precondition:

All subproducts must have the same rate times.

### Public Member Functions

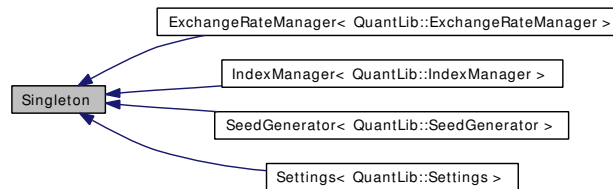
#### MarketModelMultiProduct interface

- Size **numberOfProducts** () const
- Size **maxNumberOfCashFlowsPerProductPerStep** () const
- bool **nextTimeStep** (const [CurveState](#) &currentState, std::vector< Size > &numberCashFlowsThisStep, std::vector< std::vector< [CashFlow](#) > > &cashFlowsGenerated)
- std::auto\_ptr< [MarketModelMultiProduct](#) > **clone** () const  
*returns a newly-allocated copy of itself*

## 7.709 Singleton Class Template Reference

```
#include <ql/Patterns/singleton.hpp>
```

Inheritance diagram for Singleton:



### 7.709.1 Detailed Description

```
template<class T> class QuantLib::Singleton< T >
```

Basic support for the singleton pattern.

The typical use of this class is:

```
class Foo : public Singleton<Foo> {  
    friend class Singleton<Foo>;  
private:  
    Foo() {}  
public:  
    ...  
};
```

which, albeit sub-optimal, frees one from the concerns of creating and managing the unique instance and can serve later as a single implementation point should synchronization features be added.

### Static Public Member Functions

- static T & [instance](#) ()  
*access to the unique instance*

## 7.710 SingleVariate Struct Template Reference

```
#include <ql/MonteCarlo/mctraits.hpp>
```

### 7.710.1 Detailed Description

**template<class RNG = PseudoRandom> struct QuantLib::SingleVariate< RNG >**

default Monte Carlo traits for single-variate models

**Examples:**

[DiscreteHedging.cpp](#).

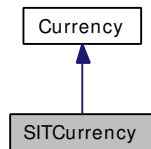
### Public Types

- enum { **allowsErrorEstimate** = RNG::allowsErrorEstimate }
- typedef RNG **rng\_traits**
- typedef [Path](#) **path\_type**
- typedef [PathPricer](#)< [path\\_type](#) > **path\_pricer\_type**
- typedef RNG::rsg\_type **rsg\_type**
- typedef [PathGenerator](#)< rsg\_type > **path\_generator\_type**

## 7.711 SITCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for SITCurrency:



### 7.711.1 Detailed Description

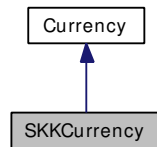
Slovenian tolar.

The ISO three-letter code is SIT; the numeric code is 705. It is divided in 100 stotinov.

## 7.712 SKKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for SKKCurrency:



### 7.712.1 Detailed Description

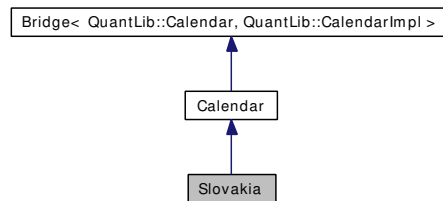
Slovak koruna.

The ISO three-letter code is SKK; the numeric code is 703. It is divided in 100 halierov.

## 7.713 Slovakia Class Reference

```
#include <ql/Calendars/slovakia.hpp>
```

Inheritance diagram for Slovakia:



### 7.713.1 Detailed Description

Slovak calendars.

Holidays for the Bratislava stock exchange (data from <http://www.bsse.sk/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- May Day, May 1st
- Liberation of the Republic, May 8th
- SS. Cyril and Methodius, July 5th
- Slovak National Uprising, August 29th
- Constitution of the Slovak Republic, September 1st
- Our Lady of the Seven Sorrows, September 15th
- All Saints Day, November 1st
- Freedom and Democracy of the Slovak Republic, November 17th
- Christmas Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th

### Public Types

- enum [Market](#) { [BSSE](#) }

## Public Member Functions

- Slovakia ([Market](#) m=BSSE)

## 7.713.2 Member Enumeration Documentation

### 7.713.2.1 enum [Market](#)

#### Enumerator:

*BSSE* Bratislava stock exchange.

## 7.714 SmileSection Class Reference

```
#include <ql/Volatilities/smilesection.hpp>
```

### 7.714.1 Detailed Description

swaption volatility smile section

This class provides the volatility smile section

### Public Member Functions

- **SmileSection** (Time expiryTime, const std::vector< Rate > &strikes, const std::vector< Rate > &volatilities)
- **SmileSection** (const std::vector< [Real](#) > &sabrParameters, const Time timeToExpiry)
- **[Real](#) variance** (const Rate &strike) const
- Volatility **volatility** (const Rate &strike) const



## 7.715 SobolRsg Class Reference

```
#include <ql/RandomNumbers/sobolrsg.hpp>
```

### 7.715.1 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 from the book "Monte Carlo Methods in Finance" by Peter Jäckel.

21 200 primitive polynomials modulo two are provided by default in QuantLib. Jäckel has calculated 8 129 334 polynomials, also available in a different file that can be downloaded from <http://quantlib.org>. If you need that many dimensions you must replace the default version of the `primitivepolynomials.c` file with the extended one.

The choice of initialization numbers (also know as free direction integers) is crucial for the homogeneity properties of the sequence. Sobol defines two homogeneity properties: Property A and Property A'.

The unit initialization numbers suggested in "Numerical Recipes in C", 2nd edition, by Press, Teukolsky, Vetterling, and Flannery (section 7.7) fail the test for Property A even for low dimensions.

Bratley and Fox published coefficients of the free direction integers up to dimension 40, crediting unpublished work of Sobol' and Levitan. See Bratley, P., Fox, B.L. (1988) "Algorithm 659: Implementing Sobol's quasirandom sequence generator," ACM Transactions on Mathematical Software 14:88-100. These values satisfy Property A for  $d \leq 20$  and  $d = 23, 31, 33, 34, 37$ ; Property A' holds for  $d \leq 6$ .

Jäckel provides in his book (section 8.3) initialization numbers up to dimension 32. Coefficients for  $d \leq 8$  are the same as in Bradley-Fox, so Property A' holds for  $d \leq 6$  but Property A holds for  $d \leq 32$ .

The implementation of Lemieux, Cieslak, and Luttmmer includes coefficients of the free direction integers up to dimension 360. Coefficients for  $d \leq 40$  are the same as in Bradley-Fox. For dimension  $40 < d \leq 360$  the coefficients have been calculated as optimal values based on the "resolution" criterion. See "RandQMC user's guide - A package for randomized quasi-Monte Carlo methods in C," by C. Lemieux, M. Cieslak, and K. Luttmmer, version January 13 2004, and references cited there (<http://www.math.ualgary.ca/~lemieux/randqmc.html>). The values up to  $d \leq 360$  has been provided to the QuantLib team by Christiane Lemieux, private communication, September 2004.

For more info on Sobol' sequences see also "Monte Carlo Methods in Financial Engineering," by P. Glasserman, 2004, Springer, section 5.2.3

#### Tests

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

### Public Types

- enum `DirectionIntegers` { `Unit`, `Jaekel`, `SobolLevitan`, `SobolLevitanLemieux` }

- typedef [Sample](#)< [Array](#) > `sample_type`

## Public Member Functions

- [SobolRsg](#) (Size dimensionality, unsigned long seed=0, DirectionIntegers directionIntegers=Jaeckel)
- void [skipTo](#) (unsigned long n)
- const std::vector< unsigned long > & [nextInt32Sequence](#) () const
- const [SobolRsg::sample\\_type](#) & [nextSequence](#) () const
- const [sample\\_type](#) & [lastSequence](#) () const
- Size [dimension](#) () const

## 7.715.2 Constructor & Destructor Documentation

- 7.715.2.1 [SobolRsg](#) (Size *dimensionality*, unsigned long *seed* = 0, DirectionIntegers *directionIntegers* = Jaeckel)

### Precondition:

dimensionality must be <= PPMT\_MAX\_DIM

## 7.715.3 Member Function Documentation

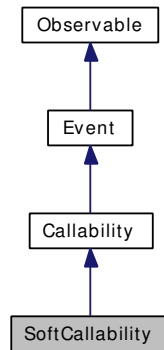
- 7.715.3.1 void [skipTo](#) (unsigned long *n*)

skip to the n-th sample in the low-discrepancy sequence

## 7.716 SoftCallability Class Reference

```
#include <ql/Instruments/convertiblebond.hpp>
```

Inheritance diagram for SoftCallability:



### 7.716.1 Detailed Description

callability leaving to the holder the possibility to convert

Examples:

[ConvertibleBonds.cpp](#).

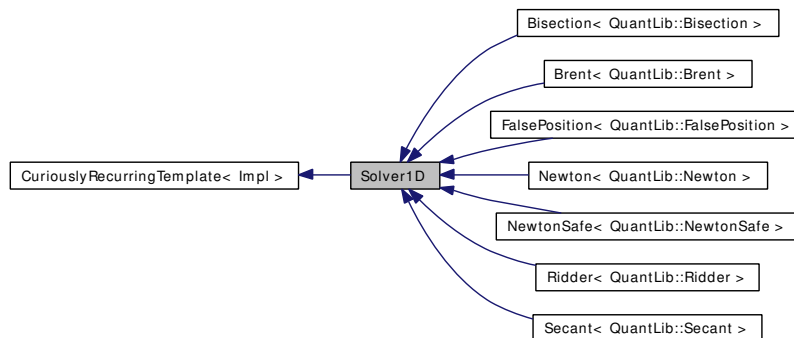
### Public Member Functions

- **SoftCallability** (const Price &price, const [Date](#) &date, Real trigger)
- Real **trigger** () const

## 7.717 Solver1D Class Template Reference

#include <ql/solver1d.hpp>

Inheritance diagram for Solver1D:



### 7.717.1 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>
    Real solveImpl(const F& f, Real 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.

#### Todo

- clean up the interface so that it is clear whether the accuracy is specified for  $x$  or  $f(x)$ .
- add target value (now the target value is 0.0)

## Public Member Functions

### Modifiers

- template<class F> Real [solve](#) (const F &f, Real accuracy, Real guess, Real step) const
- template<class F> Real [solve](#) (const F &f, Real accuracy, Real guess, Real xMin, Real xMax) const
- void [setMaxEvaluations](#) (Size evaluations)
- void [setLowerBound](#) (Real lowerBound)  
*sets the lower bound for the function domain*
- void [setUpperBound](#) (Real upperBound)  
*sets the upper bound for the function domain*

## Protected Attributes

- Real [root\\_](#)
- Real [xMin\\_](#)
- Real [xMax\\_](#)
- Real [fxMin\\_](#)
- Real [fxMax\\_](#)
- Size [maxEvaluations\\_](#)
- Size [evaluationNumber\\_](#)

## 7.717.2 Member Function Documentation

### 7.717.2.1 Real solve (const F &f, Real accuracy, Real guess, Real step) const

This method returns the zero of the function  $f$ , determined with the given accuracy  $\epsilon$ ; depending on the particular solver, this might mean that the returned  $x$  is such that  $|f(x)| < \epsilon$ , or that  $|x - \xi| < \epsilon$  where  $\xi$  is the real zero.

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.

### 7.717.2.2 Real solve (const F &f, Real accuracy, Real guess, Real xMin, Real xMax) const

This method returns the zero of the function  $f$ , determined with the given accuracy  $\epsilon$ ; depending on the particular solver, this might mean that the returned  $x$  is such that  $|f(x)| < \epsilon$ , or that  $|x - \xi| < \epsilon$  where  $\xi$  is the real zero.

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).

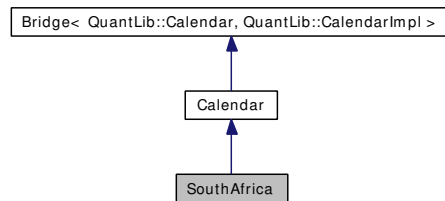
### 7.717.2.3 void setMaxEvaluations (Size evaluations)

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.

## 7.718 SouthAfrica Class Reference

```
#include <ql/Calendars/southafrica.hpp>
```

Inheritance diagram for SouthAfrica:



### 7.718.1 Detailed Description

South-African calendar.

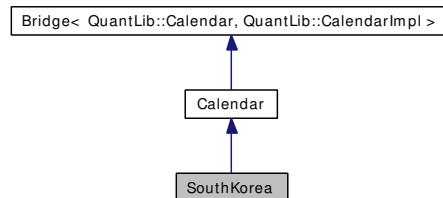
Holidays:

- 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)

## 7.719 SouthKorea Class Reference

```
#include <ql/Calendars/southkorea.hpp>
```

Inheritance diagram for SouthKorea:



### 7.719.1 Detailed Description

South Korean calendars.

Holidays for the Korea exchange (data from <http://www.kofex.com>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Independence Day, March 1st
- Arbour Day, April 5th
- Labor Day, May 1st
- Children's Day, May 5th
- Memorial Day, June 6th
- Constitution Day, July 17th
- Liberation Day, August 15th
- National Fondation Day, October 3th
- Christmas Day, December 25th

Other holidays for which no rule is given (data available for 2004-2006 only:)

- Lunar New Year
- Election Day 2004
- Buddha's birthday
- Harvest Moon Day

### Public Types

- enum [Market](#) { [KRX](#) }

## Public Member Functions

- SouthKorea ([Market](#) m=KRX)

## 7.719.2 Member Enumeration Documentation

### 7.719.2.1 enum [Market](#)

Enumerator:

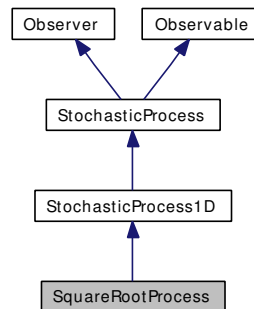
*KRX* Korea exchange.



## 7.720 SquareRootProcess Class Reference

```
#include <ql/Processes/squarerootprocess.hpp>
```

Inheritance diagram for SquareRootProcess:



### 7.720.1 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** (Real b, Real a, Volatility sigma, Real x0=0.0, const boost::shared\_ptr< discretization > &d=boost::shared\_ptr< discretization >(new [EulerDiscretization](#)))

#### StochasticProcess interface

- Real [x0](#) () const  
*returns the initial value of the state variable*
- Real [drift](#) (Time t, Real x) const  
*returns the drift part of the equation, i.e.  $\mu(t, x_t)$*
- Real [diffusion](#) (Time t, Real x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*

## 7.721 StatsHolder Class Reference

```
#include <ql/Math/gaussianstatistics.hpp>
```

### 7.721.1 Detailed Description

Helper class for precomputed distributions.

### Public Types

- typedef Real **value\_type**

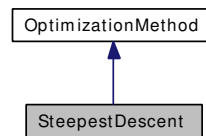
### Public Member Functions

- **StatsHolder** (Real mean, Real standardDeviation)
- Real **mean** () const
- Real **standardDeviation** () const

## 7.722 SteepestDescent Class Reference

```
#include <ql/Optimization/steepestdescent.hpp>
```

Inheritance diagram for SteepestDescent:



### 7.722.1 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 boost::shared\_ptr< [LineSearch](#) > &lineSearch)  
*default constructor*
- void [minimize](#) (const [Problem](#) &P) const  
*minimize the optimization problem P*

## 7.723 `step_iterator` Class Template Reference

```
#include <ql/Utilities/steppingiterator.hpp>
```

### 7.723.1 Detailed Description

**template<class Iterator> class QuantLib::step\_iterator< Iterator >**

Iterator advancing in constant steps.

This iterator advances an underlying random-access iterator in steps of  $n$  positions, where  $n$  is a positive integer given upon construction.

### Public Member Functions

- **step\_iterator** (const Iterator &base, Size step)
- template<class OtherIterator> **step\_iterator** (const [step\\_iterator](#)< OtherIterator > &i, typename boost::enable\_if\_convertible< OtherIterator, Iterator >::type \*=0)
- Size **step** () const
- void **increment** ()
- void **decrement** ()
- void **advance** (typename super\_t::difference\_type n)
- super\_t::difference\_type **distance\_to** (const [step\\_iterator](#) &i) const

### Related Functions

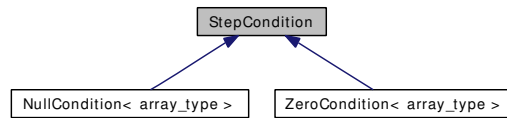
(Note that these are not member functions.)

- [step\\_iterator](#)< Iterator > [make\\_step\\_iterator](#) (Iterator it, Size step)  
*helper function to create step iterators*

## 7.724 StepCondition Class Template Reference

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

Inheritance diagram for StepCondition:



### 7.724.1 Detailed Description

```
template<class array_type> class QuantLib::StepCondition< array_type >
```

condition to be applied at every time step

#### Public Member Functions

- virtual void **applyTo** (array\_type &a, Time t) const=0

## 7.725 StepConditionSet Class Template Reference

```
#include <ql/FiniteDifferences/parallelevolver.hpp>
```

### 7.725.1 Detailed Description

```
template<typename array_type> class QuantLib::StepConditionSet< array_type >
```

Parallel evolver for multiple arrays.

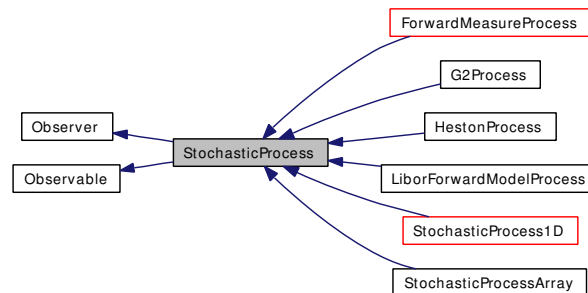
### Public Member Functions

- void **applyTo** (std::vector< array\_type > &a, [Time](#) t) const
- void **push\_back** (const itemType &a)

## 7.726 StochasticProcess Class Reference

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

Inheritance diagram for StochasticProcess:



### 7.726.1 Detailed Description

multi-dimensional stochastic process class.

This class describes a stochastic process governed by

$$dx_t = \mu(t, x_t)dt + \sigma(t, x_t) \cdot dW_t.$$

### Public Member Functions

#### Stochastic process interface

- virtual `Size` `size` () const=0  
*returns the number of dimensions of the stochastic process*
- virtual `Size` `factors` () const  
*returns the number of independent factors of the process*
- virtual `Disposable`< `Array` > `initialValues` () const=0  
*returns the initial values of the state variables*
- virtual `Disposable`< `Array` > `drift` (Time t, const `Array` &x) const=0  
*returns the drift part of the equation, i.e.,  $\mu(t, x_t)$*
- virtual `Disposable`< `Matrix` > `diffusion` (Time t, const `Array` &x) const=0  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*
- virtual `Disposable`< `Array` > `expectation` (Time t0, const `Array` &x0, Time dt) const
- virtual `Disposable`< `Matrix` > `stdDeviation` (Time t0, const `Array` &x0, Time dt) const
- virtual `Disposable`< `Matrix` > `covariance` (Time t0, const `Array` &x0, Time dt) const
- virtual `Disposable`< `Array` > `evolve` (Time t0, const `Array` &x0, Time dt, const `Array` &dw) const
- virtual `Disposable`< `Array` > `apply` (const `Array` &x0, const `Array` &dx) const

#### utilities

- virtual Time [time](#) (const [Date](#) &) const

#### Observer interface

- void [update](#) ()

### Protected Member Functions

- [StochasticProcess](#) (const boost::shared\_ptr< [discretization](#) > &)

### Protected Attributes

- boost::shared\_ptr< [discretization](#) > [discretization\\_](#)

### Classes

- class [discretization](#)  
*discretization of a stochastic process over a given time interval*

## 7.726.2 Member Function Documentation

### 7.726.2.1 virtual [Disposable](#)<[Array](#)> expectation (Time *t0*, const [Array](#) & *x0*, Time *dt*) const [virtual]

returns the expectation  $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [G2Process](#), [G2ForwardProcess](#), and [StochasticProcessArray](#).

### 7.726.2.2 virtual [Disposable](#)<[Matrix](#)> stdDeviation (Time *t0*, const [Array](#) & *x0*, Time *dt*) const [virtual]

returns the standard deviation  $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [G2Process](#), [G2ForwardProcess](#), and [StochasticProcessArray](#).

### 7.726.2.3 virtual [Disposable](#)<[Matrix](#)> covariance (Time *t0*, const [Array](#) & *x0*, Time *dt*) const [virtual]

returns the covariance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [G2Process](#), [G2ForwardProcess](#), [LiborForwardModelProcess](#), and [StochasticProcessArray](#).



**7.726.2.4** `virtual Disposable<Array> evolve (Time t0, const Array & x0, Time dt, const Array & dw) const` [virtual]

returns the asset value after a time interval  $\Delta t$  according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where  $E$  is the expectation and  $S$  the standard deviation.

Reimplemented in [LiborForwardModelProcess](#).

**7.726.2.5** `virtual Disposable<Array> apply (const Array & x0, const Array & dx) const` [virtual]

applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented in [HestonProcess](#), [LiborForwardModelProcess](#), and [StochasticProcessArray](#).

**7.726.2.6** `virtual Time time (const Date &) const` [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

**Note:**

As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented in [GeneralizedBlackScholesProcess](#), [HestonProcess](#), [Merton76Process](#), and [StochasticProcessArray](#).

**7.726.2.7** `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.

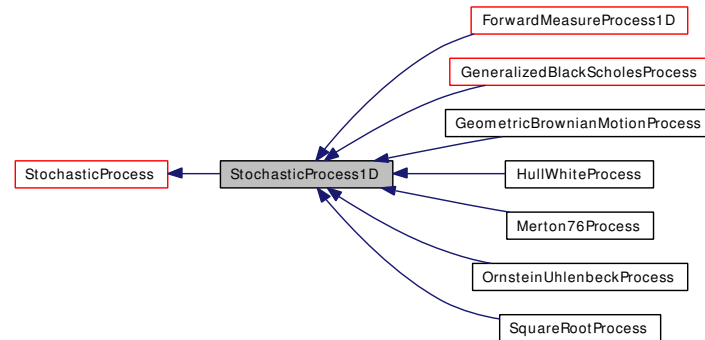
Implements [Observer](#).

Reimplemented in [GeneralizedBlackScholesProcess](#).

## 7.727 StochasticProcess1D Class Reference

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

Inheritance diagram for StochasticProcess1D:



### 7.727.1 Detailed Description

1-dimensional stochastic process

This class describes a stochastic process governed by

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

### Public Member Functions

#### 1-D stochastic process interface

- virtual Real [x0](#) () const=0  
*returns the initial value of the state variable*
- virtual Real [drift](#) (Time t, Real x) const=0  
*returns the drift part of the equation, i.e.  $\mu(t, x_t)$*
- virtual Real [diffusion](#) (Time t, Real x) const=0  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*
- virtual Real [expectation](#) (Time t0, Real x0, Time dt) const
- virtual Real [stdDeviation](#) (Time t0, Real x0, Time dt) const
- virtual Real [variance](#) (Time t0, Real x0, Time dt) const
- virtual Real [evolve](#) (Time t0, Real x0, Time dt, Real dw) const
- virtual Real [apply](#) (Real x0, Real dx) const

### Protected Member Functions

- StochasticProcess1D (const boost::shared\_ptr< [discretization](#) > &)

### Protected Attributes

- boost::shared\_ptr< [discretization](#) > [discretization\\_](#)

## Classes

- class [discretization](#)  
*discretization of a 1-D stochastic process*

## 7.727.2 Member Function Documentation

### 7.727.2.1 virtual Real expectation (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

returns the expectation  $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [HullWhiteProcess](#), [HullWhiteForwardProcess](#), and [OrnsteinUhlenbeckProcess](#).

### 7.727.2.2 virtual Real stdDeviation (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

returns the standard deviation  $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [HullWhiteProcess](#), [HullWhiteForwardProcess](#), and [OrnsteinUhlenbeckProcess](#).

### 7.727.2.3 virtual Real variance (Time $t_0$ , Real $x_0$ , Time $dt$ ) const [virtual]

returns the variance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [HullWhiteProcess](#), [HullWhiteForwardProcess](#), and [OrnsteinUhlenbeckProcess](#).

### 7.727.2.4 virtual Real evolve (Time $t_0$ , Real $x_0$ , Time $dt$ , Real $dw$ ) const [virtual]

returns the asset value after a time interval  $\Delta t$  according to the given discretization. By default, it returns

$$E(x_0, t_0, \Delta t) + S(x_0, t_0, \Delta t) \cdot \Delta w$$

where  $E$  is the expectation and  $S$  the standard deviation.

### 7.727.2.5 virtual Real apply (Real $x_0$ , Real $dx$ ) const [virtual]

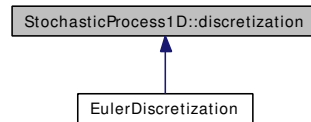
applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented in [GeneralizedBlackScholesProcess](#), and [Merton76Process](#).

## 7.728 StochasticProcess1D::discretization Class Reference

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

Inheritance diagram for StochasticProcess1D::discretization:



### 7.728.1 Detailed Description

discretization of a 1-D stochastic process

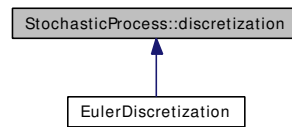
#### Public Member Functions

- virtual Real **drift** (const [StochasticProcess1D](#) &, Time t0, Real x0, Time dt) const=0
- virtual Real **diffusion** (const [StochasticProcess1D](#) &, Time t0, Real x0, Time dt) const=0
- virtual Real **variance** (const [StochasticProcess1D](#) &, Time t0, Real x0, Time dt) const=0

## 7.729 StochasticProcess::discretization Class Reference

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

Inheritance diagram for StochasticProcess::discretization:



### 7.729.1 Detailed Description

discretization of a stochastic process over a given time interval

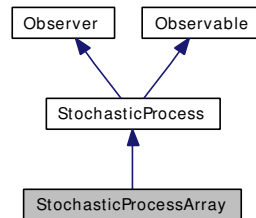
#### Public Member Functions

- virtual [Disposable](#)< [Array](#) > **drift** (const [StochasticProcess](#) &, Time t0, const [Array](#) &x0, Time dt) const=0
- virtual [Disposable](#)< [Matrix](#) > **diffusion** (const [StochasticProcess](#) &, Time t0, const [Array](#) &x0, Time dt) const=0
- virtual [Disposable](#)< [Matrix](#) > **covariance** (const [StochasticProcess](#) &, Time t0, const [Array](#) &x0, Time dt) const=0

## 7.730 StochasticProcessArray Class Reference

```
#include <ql/Processes/stochasticprocessarray.hpp>
```

Inheritance diagram for StochasticProcessArray:



### 7.730.1 Detailed Description

Array of correlated 1-D stochastic processes

#### Public Member Functions

- **StochasticProcessArray** (const std::vector< boost::shared\_ptr< [StochasticProcess1D](#) > > &, const [Matrix](#) &correlation)
- Size [size](#) () const  
*returns the number of dimensions of the stochastic process*
- [Disposable](#)< [Array](#) > [initialValues](#) () const  
*returns the initial values of the state variables*
- [Disposable](#)< [Array](#) > [drift](#) (Time t, const [Array](#) &x) const  
*returns the drift part of the equation, i.e.,  $\mu(t, x_t)$*
- [Disposable](#)< [Matrix](#) > [diffusion](#) (Time t, const [Array](#) &x) const  
*returns the diffusion part of the equation, i.e.  $\sigma(t, x_t)$*
- [Disposable](#)< [Array](#) > [expectation](#) (Time t0, const [Array](#) &x0, Time dt) const
- [Disposable](#)< [Matrix](#) > [stdDeviation](#) (Time t0, const [Array](#) &x0, Time dt) const
- [Disposable](#)< [Matrix](#) > [covariance](#) (Time t0, const [Array](#) &x0, Time dt) const
- [Disposable](#)< [Array](#) > [apply](#) (const [Array](#) &x0, const [Array](#) &dx) const
- Time [time](#) (const [Date](#) &) const
- const boost::shared\_ptr< [StochasticProcess1D](#) > & [process](#) (Size i) const
- [Disposable](#)< [Matrix](#) > [correlation](#) () const

#### Protected Attributes

- std::vector< boost::shared\_ptr< [StochasticProcess1D](#) > > [processes\\_](#)
- [Matrix](#) [sqrtCorrelation\\_](#)

## 7.730.2 Member Function Documentation

### 7.730.2.1 `Disposable<Array> expectation (Time $t_0$ , const Array & $x_0$ , Time $dt$ ) const` [virtual]

returns the expectation  $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

### 7.730.2.2 `Disposable<Matrix> stdDeviation (Time $t_0$ , const Array & $x_0$ , Time $dt$ ) const` [virtual]

returns the standard deviation  $S(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

### 7.730.2.3 `Disposable<Matrix> covariance (Time $t_0$ , const Array & $x_0$ , Time $dt$ ) const` [virtual]

returns the covariance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

### 7.730.2.4 `Disposable<Array> apply (const Array & $x_0$ , const Array & $dx$ ) const` [virtual]

applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented from [StochasticProcess](#).

### 7.730.2.5 `Time time (const Date &) const` [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

#### Note:

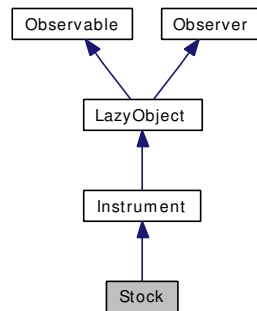
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

## 7.731 Stock Class Reference

```
#include <ql/Instruments/stock.hpp>
```

Inheritance diagram for Stock:



### 7.731.1 Detailed Description

Simple stock class.

#### Public Member Functions

- **Stock** (const [Handle](#)< [Quote](#) > &quote)
- **bool isExpired** () const  
*returns whether the instrument is still tradable.*

#### Protected Member Functions

- **void performCalculations** () const

### 7.731.2 Member Function Documentation

#### 7.731.2.1 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.

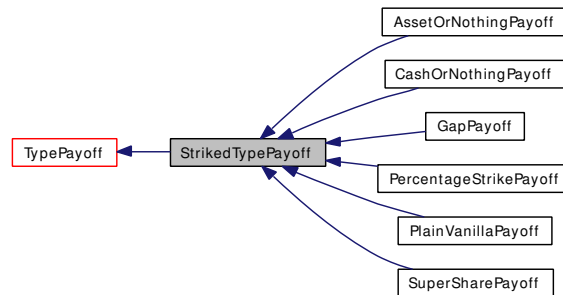
Reimplemented from [Instrument](#).



## 7.732 StrikedTypePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for StrikedTypePayoff:



### 7.732.1 Detailed Description

Intermediate class for payoffs based on a fixed strike.

#### Public Member Functions

- `StrikedTypePayoff` (Option::Type type, Real strike)
- Real `strike` () const

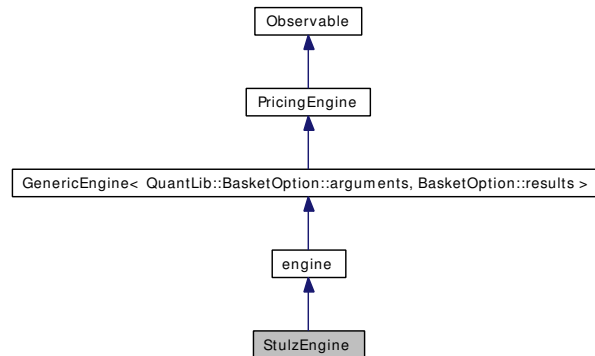
#### Protected Attributes

- Real `strike_`

## 7.733 StulzEngine Class Reference

```
#include <ql/PricingEngines/Basket/stulzengine.hpp>
```

Inheritance diagram for StulzEngine:



### 7.733.1 Detailed Description

Pricing engine for 2D European Baskets.

This class implements formulae from "Options on the Minimum or the Maximum of Two Risky Assets", Rene Stulz, Journal of Financial Economics (1982) 10, 161-185.

#### Tests

the correctness of the returned value is tested by reproducing results available in literature.

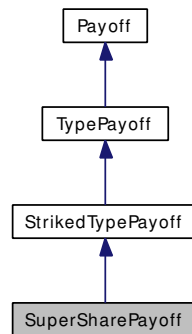
### Public Member Functions

- void **calculate** () const

## 7.734 SuperSharePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for SuperSharePayoff:



### 7.734.1 Detailed Description

Binary supershare payoff.

#### Public Member Functions

- **SuperSharePayoff** (Option::Type type, Real strike, Real strikeIncrement)
- Real **operator()** (Real price) const
- Real **strikeIncrement** () const
- virtual void **accept** ([AcyclicVisitor](#) &)

## 7.735 SVD Class Reference

```
#include <ql/Math/svd.hpp>
```

### 7.735.1 Detailed Description

Singular value decomposition.

Refer to Golub and Van Loan: [Matrix](#) computation, The Johns Hopkins University Press

#### Tests

the correctness of the returned values is tested by checking their properties.

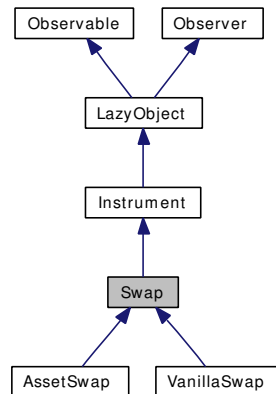
### Public Member Functions

- **SVD** (const [Matrix](#) &)
- const [Matrix](#) & **U** () const
- const [Matrix](#) & **V** () const
- const [Array](#) & **singularValues** () const
- [Disposable](#)< [Matrix](#) > **S** () const
- [Real](#) **norm2** ()
- [Real](#) **cond** ()
- Integer **rank** ()
- [Disposable](#)< [Array](#) > **solveFor** (const [Array](#) &) const

## 7.736 Swap Class Reference

```
#include <ql/Instruments/swap.hpp>
```

Inheritance diagram for Swap:



### 7.736.1 Detailed Description

Interest rate swap.

The cash flows belonging to the first leg are paid; the ones belonging to the second leg are received.

### Public Member Functions

- **Swap** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, const Leg &firstLeg, const Leg &secondLeg)
- **Swap** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, const std::vector< Leg > &legs, const std::vector< bool > &payer)

#### Instrument interface

- bool [isExpired](#) () const  
*returns whether the instrument is still tradable.*

#### Additional interface

- [Date](#) [startDate](#) () const
- [Date](#) [maturity](#) () const
- [Real](#) [legBPS](#) (Size j) const
- [Real](#) [legNPV](#) (Size j) const
- const Leg & [leg](#) (Size j) const

### Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

## Protected Attributes

- [Handle](#)< [YieldTermStructure](#) > `termStructure_`
- `std::vector< Leg > legs_`
- `std::vector< Real > payer_`
- `std::vector< Real > legNPV_`
- `std::vector< Real > legBPS_`

## 7.736.2 Constructor & Destructor Documentation

**7.736.2.1** [Swap](#) (const [Handle](#)< [YieldTermStructure](#) > & *termStructure*, const Leg & *firstLeg*, const Leg & *secondLeg*)

The cash flows belonging to the first leg are paid; the ones belonging to the second leg are received.

**7.736.2.2** [Swap](#) (const [Handle](#)< [YieldTermStructure](#) > & *termStructure*, const std::vector< Leg > & *legs*, const std::vector< bool > & *payer*)

Multi leg constructor.

## 7.736.3 Member Function Documentation

**7.736.3.1** `void setupExpired () const` [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met. Reimplemented from [Instrument](#).

**7.736.3.2** `void performCalculations () const` [protected, virtual]

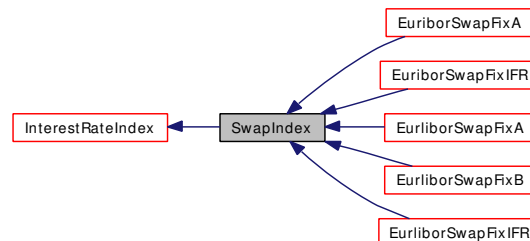
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

## 7.737 SwapIndex Class Reference

```
#include <ql/Indexes/swapindex.hpp>
```

Inheritance diagram for SwapIndex:



### 7.737.1 Detailed Description

base class for swap-rate indexes

### Public Member Functions

- **SwapIndex** (const std::string &familyName, Integer years, Integer settlementDays, const [Currency](#) &currency, const [Calendar](#) &calendar, [Frequency](#) fixedLegFrequency, [BusinessDayConvention](#) fixedLegConvention, const [DayCounter](#) &fixedLegDayCounter, const boost::shared\_ptr<[Xibor](#)> &iborIndex)

#### InterestRateIndex interface

- boost::shared\_ptr<[YieldTermStructure](#)> **termStructure** () const
- [Rate](#) forecastFixing (const [Date](#) &fixingDate) const

#### Inspectors

- [Frequency](#) fixedLegFrequency () const
- [BusinessDayConvention](#) fixedLegConvention () const
- boost::shared\_ptr<[Xibor](#)> **iborIndex** () const
- [Schedule](#) fixedRateSchedule (const [Date](#) &fixingDate) const
- boost::shared\_ptr<[VanillaSwap](#)> **underlyingSwap** (const [Date](#) &fixingDate) const

### Protected Attributes

- Integer years\_
- boost::shared\_ptr<[Xibor](#)> **iborIndex\_**
- [Frequency](#) fixedLegFrequency\_
- [BusinessDayConvention](#) fixedLegConvention\_

## 7.737.2 Member Function Documentation

7.737.2.1 `boost::shared_ptr<VanillaSwap> underlyingSwap (const Date & fixingDate) const`

### [Warning](#)

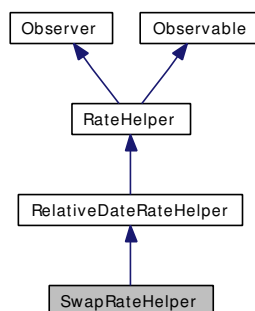
Relinking the term structure underlying the index will not have effect on the returned swap.



## 7.738 SwapRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for SwapRateHelper:



### 7.738.1 Detailed Description

Rate helper for bootstrapping over swap rates.

Examples:

[swapvaluation.cpp](#).

### Public Member Functions

- **SwapRateHelper** (const [Handle](#)< [Quote](#) > &rate, const [Period](#) &tenor, Integer settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, const boost::shared\_ptr< [Xibor](#) > &index)
- **SwapRateHelper** ([Rate](#) rate, const [Period](#) &tenor, Integer settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, const boost::shared\_ptr< [Xibor](#) > &index)
- **Real impliedQuote** () const
- void **setTermStructure** ([YieldTermStructure](#) \*)  
*sets the term structure to be used for pricing*

### Protected Member Functions

- void **initializeDates** ()

### Protected Attributes

- [Period](#) tenor\_
- Integer settlementDays\_
- [Calendar](#) calendar\_
- [BusinessDayConvention](#) fixedConvention\_

- [Frequency](#) `fixedFrequency_`
- [DayCounter](#) `fixedDayCount_`
- `boost::shared_ptr< Xibor > index_`
- `boost::shared_ptr< VanillaSwap > swap_`
- `Handle< YieldTermStructure > termStructureHandle_`

## 7.738.2 Member Function Documentation

7.738.2.1 `void setTermStructure (YieldTermStructure *)` [virtual]

sets the term structure to be used for pricing

### Warning

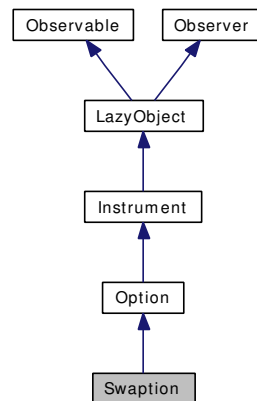
Being a pointer and not a `shared_ptr`, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

## 7.739 Swaption Class Reference

```
#include <ql/Instruments/swaption.hpp>
```

Inheritance diagram for Swaption:



### 7.739.1 Detailed Description

Swaption class

#### Tests

- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption decreases (resp. increases) with the strike.
- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption increases (resp. decreases) with the spread.
- the correctness of the returned value is tested by checking it against that of a swaption on a swap with no spread and a correspondingly adjusted fixed rate.
- the correctness of the returned value is tested by checking it against a known good value.
- the correctness of the returned value of cash settled swaptions is tested by checking the modified annuity against a value calculated without using the [Swaption](#) class.

#### Todo

add greeks and explicit exercise lag

#### Examples:

[BermudanSwaption.cpp](#).

### Public Member Functions

- **Swaption** (const boost::shared\_ptr< [VanillaSwap](#) > &swap, const boost::shared\_ptr< [Exercise](#) > &exercise, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared\_ptr< [PricingEngine](#) > &engine, Settlement::Type delivery=Settlement::Physical)
- void [setupArguments](#) ([Arguments](#) \*) const

- [Volatility impliedVolatility](#) ([Real](#) price, [Real](#) accuracy=1.0e-4, Size maxEvaluations=100, Volatility minVol=QL\_MIN\_VOLATILITY, Volatility maxVol=QL\_MAX\_VOLATILITY) const

*implied volatility*

#### Instrument interface

- bool [isExpired](#) () const  
*returns whether the instrument is still tradable.*

#### Inspectors

- Settlement::Type [settlementType](#) () const
- const boost::shared\_ptr< [VanillaSwap](#) > & [underlyingSwap](#) () const

### Classes

- class [arguments](#)  
*Arguments for swaption calculation*
- class [engine](#)  
*base class for swaption engines*
- class [results](#)  
*Results from swaption calculation*

## 7.739.2 Member Function Documentation

### 7.739.2.1 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](#).

## 7.740 Swaption::arguments Class Reference

```
#include <ql/Instruments/swaption.hpp>
```

### 7.740.1 Detailed Description

Arguments for swaption calculation

#### Public Member Functions

- void **validate** () const

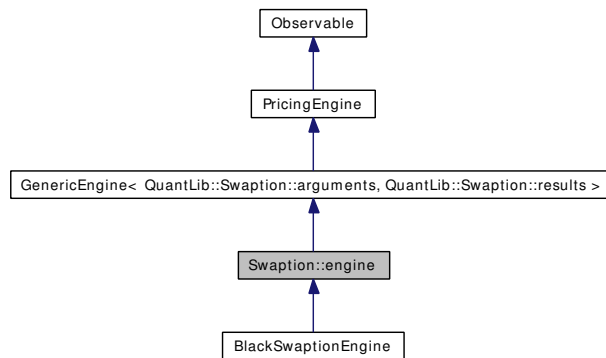
#### Public Attributes

- Rate **fairRate**
- Rate **fixedRate**
- Real **fixedBPS**
- Real **fixedCashBPS**
- Settlement::Type **settlementType**

## 7.741 Swaption::engine Class Reference

```
#include <ql/Instruments/swaption.hpp>
```

Inheritance diagram for Swaption::engine:



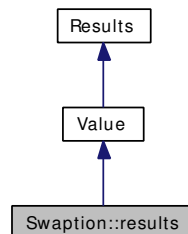
### 7.741.1 Detailed Description

base class for swaption engines

## 7.742 Swaption::results Class Reference

```
#include <ql/Instruments/swaption.hpp>
```

Inheritance diagram for Swaption::results:



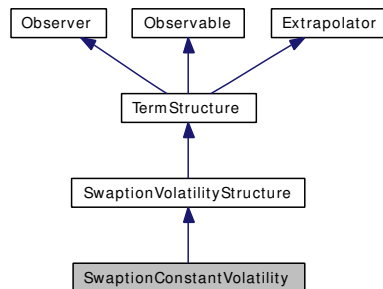
### 7.742.1 Detailed Description

Results from swaption calculation

## 7.743 SwaptionConstantVolatility Class Reference

#include <ql/Volatilities/swaptionconstantvol.hpp>

Inheritance diagram for SwaptionConstantVolatility:



### 7.743.1 Detailed Description

Constant swaption volatility, no time-strike dependence.

#### SwaptionConstantVolatility interface

- [Date](#) [maxStartDate](#) () const  
*the latest start date for which the term structure can return vols*
- [Time](#) [maxStartTime](#) () const  
*the latest start time for which the term structure can return vols*
- [Period](#) [maxLength](#) () const  
*the largest length for which the term structure can return vols*
- [Time](#) [maxTimeLength](#) () const  
*the largest length for which the term structure can return vols*
- [Real](#) [minStrike](#) () const  
*the minimum strike for which the term structure can return vols*
- [Real](#) [maxStrike](#) () const  
*the maximum strike for which the term structure can return vols*
- [boost::shared\\_ptr](#)< [SmileSection](#) > [smileSection](#) (const [Date](#) &start, const [Period](#) &length) const  
*return trivial smile section*
- [Volatility](#) [volatilityImpl](#) ([Time](#), [Time](#), [Rate](#)) const  
*implements the actual volatility calculation in derived classes*
- [boost::shared\\_ptr](#)< [SmileSection](#) > [smileSection](#) ([Time](#) start, [Time](#) length) const  
*return smile section*



- Volatility **volatilityImpl** (const [Date](#) &, const [Period](#) &, Rate) const

## Public Member Functions

- **SwaptionConstantVolatility** (const [Date](#) &referenceDate, Volatility volatility, const [DayCounter](#) &dayCounter)
- **SwaptionConstantVolatility** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- **SwaptionConstantVolatility** (Integer settlementDays, const [Calendar](#) &, Volatility volatility, const [DayCounter](#) &dayCounter)
- **SwaptionConstantVolatility** (Integer settlementDays, const [Calendar](#) &, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)

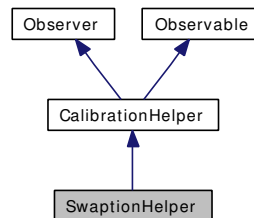
## TermStructure interface

- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*

## 7.744 SwaptionHelper Class Reference

```
#include <ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp>
```

Inheritance diagram for SwaptionHelper:



### 7.744.1 Detailed Description

calibration helper for ATM swaption

Examples:

[BermudanSwaption.cpp](#).

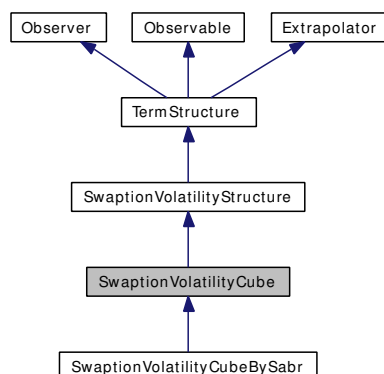
### Public Member Functions

- **SwaptionHelper** (const [Period](#) &maturity, const [Period](#) &length, const [Handle](#)< [Quote](#) > &volatility, const boost::shared\_ptr< [Xibor](#) > &index, [Frequency](#) fixedLegFrequency, const [DayCounter](#) &fixedLegDayCounter, const [DayCounter](#) &floatingLegDayCounter, const [Handle](#)< [YieldTermStructure](#) > &termStructure, bool calibrateVolatility=false)
- **SwaptionHelper** (const [Period](#) &maturity, const [Period](#) &length, const [Handle](#)< [Quote](#) > &volatility, const boost::shared\_ptr< [Xibor](#) > &index, const [Period](#) &fixedLegTenor, const [DayCounter](#) &fixedLegDayCounter, const [DayCounter](#) &floatingLegDayCounter, const [Handle](#)< [YieldTermStructure](#) > &termStructure, bool calibrateVolatility=false)
- virtual void **addTimesTo** (std::list< Time > &times) const
- virtual [Real](#) **modelValue** () const  
*returns the price of the instrument according to the model*
- virtual [Real](#) **blackPrice** (Volatility volatility) const  
*Black price given a volatility.*

## 7.745 SwaptionVolatilityCube Class Reference

```
#include <ql/Volatilities/swaptionvolcube.hpp>
```

Inheritance diagram for SwaptionVolatilityCube:



### 7.745.1 Detailed Description

#### Warning

this class is not finalized and its interface might change in subsequent releases.

### Public Member Functions

- **SwaptionVolatilityCube** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &atmVolStructure, const std::vector< [Period](#) > &expiries, const std::vector< [Period](#) > &lengths, const std::vector< [Spread](#) > &strikeSpreads, const [Calendar](#) &calendar, Integer swapSettlementDays, [Frequency](#) fixedLegFrequency, [BusinessDayConvention](#) fixedLegConvention, const [DayCounter](#) &fixedLegDayCounter, const boost::shared\_ptr< [Xibor](#) > &iborIndex, Time shortTenor=2, const boost::shared\_ptr< [Xibor](#) > &iborIndexShortTenor=boost::shared\_ptr< [Xibor](#) >())
- Rate **atmStrike** (const [Period](#) &optionTenor, const [Period](#) &swapTenor) const
- Rate **atmStrike** (const [Date](#) &optionDate, const [Period](#) &swapTenor) const

#### TermStructure interface

- const [Date](#) & **referenceDate** () const  
*the date at which discount = 1.0 and/or variance = 0.0*
- [DayCounter](#) **dayCounter** () const  
*the day counter used for date/time conversion*

#### SwaptionVolatilityStructure interface

- [Date](#) **maxStartDate** () const  
*the latest start date for which the term structure can return vols*

- Time [maxStartTime](#) () const  
*the latest start time for which the term structure can return vols*
- Period [maxLength](#) () const  
*the largest length for which the term structure can return vols*
- Time [maxTimeLength](#) () const  
*the largest length for which the term structure can return vols*
- Rate [minStrike](#) () const  
*the minimum strike for which the term structure can return vols*
- Rate [maxStrike](#) () const  
*the maximum strike for which the term structure can return vols*

## Protected Member Functions

### SwaptionVolatilityStructure interface

- `std::pair< Time, Time > convertDates (const Date &exerciseDate, const Period &length)`  
const  
*implements the conversion between dates and times*

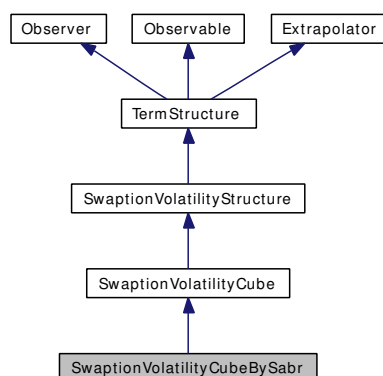
## Protected Attributes

- [Handle](#)< [SwaptionVolatilityStructure](#) > [atmVolStructure\\_](#)
- `std::vector< Date > exerciseDates\_`
- `std::vector< Time > exerciseTimes\_`
- `std::vector< Real > exerciseDatesAsReal\_`
- [LinearInterpolation](#) [exerciseInterpolator\\_](#)
- `std::vector< Period > lengths\_`
- `std::vector< Time > timeLengths\_`
- Size [nExercise\\_](#)
- Size [nlengths\\_](#)
- Size [nStrikes\\_](#)
- `std::vector< Spread > strikeSpreads\_`
- `std::vector< Rate > localStrikes\_`
- `std::vector< Volatility > localSmile\_`
- Integer [swapSettlementDays\\_](#)
- [Frequency](#) [fixedLegFrequency\\_](#)
- [BusinessDayConvention](#) [fixedLegConvention\\_](#)
- [DayCounter](#) [fixedLegDayCounter\\_](#)
- `boost::shared_ptr< Xibor > iborIndex\_`
- Time [shortTenor\\_](#)
- `boost::shared_ptr< Xibor > iborIndexShortTenor\_`

## 7.746 SwaptionVolatilityCubeBySabr Class Reference

```
#include <ql/Volatilities/swaptionvolcubebySabr.hpp>
```

Inheritance diagram for SwaptionVolatilityCubeBySabr:



### 7.746.1 Detailed Description

#### Warning

this class is not finalized and its interface might change in subsequent releases.

#### Public Member Functions

- **SwaptionVolatilityCubeBySabr** (const [Handle](#)< [SwaptionVolatilityStructure](#) > &atmVolStructure, const std::vector< [Period](#) > &expiries, const std::vector< [Period](#) > &lengths, const std::vector< [Spread](#) > &strikeSpreads, const std::vector< std::vector< [Handle](#)< [Quote](#) > > > &volSpreads, const [Calendar](#) &calendar, [Integer](#) swapSettlementDays, [Frequency](#) fixedLegFrequency, [BusinessDayConvention](#) fixedLegConvention, const [DayCounter](#) &fixedLegDayCounter, const boost::shared\_ptr< [Xibor](#) > &iborIndex, Time shortTenor, const boost::shared\_ptr< [Xibor](#) > &iborIndexShortTenor, const [Matrix](#) &parameters-Guess, std::vector< bool > isParameterFixed, bool isAtmCalibrated)
- const [Matrix](#) & **marketVolCube** (Size i) const
- void **recalibration** ([Real](#) beta)
- [Matrix](#) **sparseSabrParameters** () const
- [Matrix](#) **denseSabrParameters** () const
- [Matrix](#) **marketVolCube** () const
- [Matrix](#) **volCubeAtmCalibrated** () const
- boost::shared\_ptr< [SmileSection](#) > **smileSection** (const [Date](#) &exerciseDate, const [Period](#) &length) const
- boost::shared\_ptr< [SmileSection](#) > **smileSection** (Time start, Time length) const  
*return smile section*

#### Protected Member Functions

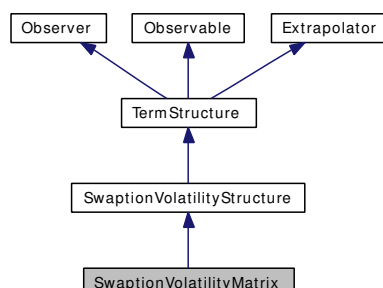
- boost::shared\_ptr< [SmileSection](#) > **smileSection** (Time start, Time length, const Cube &sabrParametersCube) const

- [Volatility](#) **volatilityImpl** (Time start, Time length, Rate strike) const  
*implements the actual volatility calculation in derived classes*
- [Volatility](#) **volatilityImpl** (const [Date](#) &exerciseDate, const [Period](#) &length, Rate strike) const
- Cube **sabrCalibration** (const Cube &marketVolCube) const
- void **fillVolatilityCube** ()
- void **createSparseSmiles** ()
- std::vector< [Real](#) > **spreadVolInterpolation** (const [Date](#) &atmExerciseDate, const [Period](#) &atmSwapTenor)

## 7.747 SwaptionVolatilityMatrix Class Reference

```
#include <ql/Volatilities/swaptionvolmatrix.hpp>
```

Inheritance diagram for SwaptionVolatilityMatrix:



### 7.747.1 Detailed Description

At-the-money swaption-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.

The volatility matrix  $M$  must be defined so that:

- the number of rows equals the number of exercise dates;
- the number of columns equals the number of swap tenors;
- $M[i][j]$  contains the volatility corresponding to the  $i$ -th exercise and  $j$ -th tenor.

### Public Member Functions

- **SwaptionVolatilityMatrix** (const std::vector< [Period](#) > &expiries, const [Calendar](#) &calendar, const [BusinessDayConvention](#) bdc, const std::vector< [Period](#) > &tenors, const std::vector< std::vector< [Handle](#)< [Quote](#) > > &vols, const [DayCounter](#) &dayCounter)
- **SwaptionVolatilityMatrix** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &exerciseDates, const std::vector< [Period](#) > &lengths, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter)
- **SwaptionVolatilityMatrix** (const std::vector< [Date](#) > &exerciseDates, const std::vector< [Period](#) > &tenors, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter)
- **SwaptionVolatilityMatrix** (const std::vector< [Period](#) > &expiries, const [Calendar](#) &calendar, const [BusinessDayConvention](#) bdc, const std::vector< [Period](#) > &tenors, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter)
- const std::vector< [Date](#) > & **exerciseDates** () const
- const std::vector< [Time](#) > & **exerciseTimes** () const
- const std::vector< [Period](#) > & **lengths** () const
- const std::vector< [Time](#) > & **timeLengths** () const

### TermStructure interface

- [DayCounter dayCounter](#) () const  
*the day counter used for date/time conversion*

### SwaptionVolatilityStructure interface

- [Date maxStartDate](#) () const  
*the latest start date for which the term structure can return vols*
- [Time maxStartTime](#) () const  
*the latest start time for which the term structure can return vols*
- [Period maxLength](#) () const  
*the largest length for which the term structure can return vols*
- [Time maxTimeLength](#) () const  
*the largest length for which the term structure can return vols*
- [Rate minStrike](#) () const  
*the minimum strike for which the term structure can return vols*
- [Rate maxStrike](#) () const  
*the maximum strike for which the term structure can return vols*
- [boost::shared\\_ptr< SmileSection > smileSection](#) (const [Date](#) &exerciseDate, const [Period](#) &length) const  
*return trivial smile section*
- [virtual boost::shared\\_ptr< SmileSection > smileSection](#) (Time start, Time length) const  
*return trivial smile section*
- [std::pair< Time, Time > convertDates](#) (const [Date](#) &exerciseDate, const [Period](#) &length) const  
*implements the conversion between dates and times*

### Other inspectors

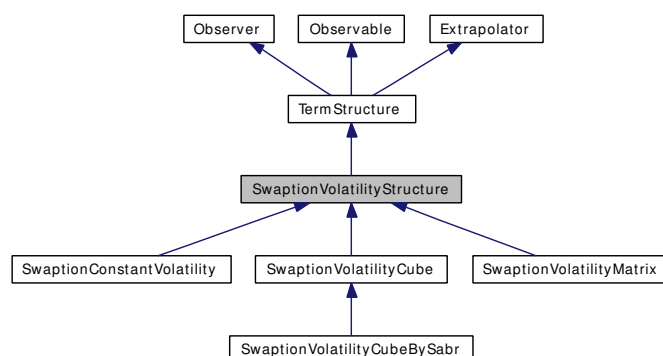
- [std::pair< Size, Size > locate](#) (const [Date](#) &exerciseDate, const [Period](#) &length) const  
*returns the lower indexes of surrounding volatility matrix corners*
- [std::pair< Size, Size > locate](#) (Time exerciseTime, Time length) const  
*returns the lower indexes of surrounding volatility matrix corners*



## 7.748 SwaptionVolatilityStructure Class Reference

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

Inheritance diagram for SwaptionVolatilityStructure:



### 7.748.1 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

- [Date](#) **maxDate** () const  
*the latest date for which the curve can return values*
- [Time](#) **maxTime** () const  
*the latest time for which the curve can return values*
- virtual [boost::shared\\_ptr](#)< [SmileSection](#) > **smileSection** (const [Date](#) &start, const [Period](#) &length) const
- virtual [std::pair](#)< [Time](#), [Time](#) > **convertDates** (const [Date](#) &exerciseDate, const [Period](#) &length) const  
*implements the conversion between dates and times*

### Volatility and Variance

- [Volatility](#) **volatility** (const [Period](#) &optionTenor, const [Period](#) &swapTenor, [Rate](#) strike, bool extrapolate=false) const  
*returns the volatility for a given starting date and length*
- [Volatility](#) **volatility** (const [Date](#) &exerciseDate, const [Period](#) &swapTenor, [Rate](#) strike, bool extrapolate=false) const  
*returns the volatility for a given starting date and length*

- **Real blackVariance** (const [Date](#) &exerciseDate, const [Period](#) &swapTenor, Rate strike, bool extrapolate=false) const  
*returns the Black variance for a given starting date and length*
- **Volatility volatility** (Time exerciseTime, Time length, Rate strike, bool extrapolate=false) const  
*returns the volatility for a given starting time and length*
- **Real blackVariance** (Time exerciseTime, Time length, Rate strike, bool extrapolate=false) const  
*returns the Black variance for a given starting time and length*

### Limits

- virtual [Date](#) **maxStartDate** () const=0  
*the latest start date for which the term structure can return vols*
- virtual [Period](#) **maxLength** () const=0  
*the largest length for which the term structure can return vols*
- virtual Time **maxStartTime** () const  
*the latest start time for which the term structure can return vols*
- virtual Time **maxTimeLength** () const  
*the largest length for which the term structure can return vols*
- virtual Rate **minStrike** () const=0  
*the minimum strike for which the term structure can return vols*
- virtual Rate **maxStrike** () const=0  
*the maximum strike for which the term structure can return vols*

### Protected Member Functions

- virtual boost::shared\_ptr< [SmileSection](#) > **smileSection** (Time start, Time length) const=0  
*return smile section*
- virtual [Volatility](#) **volatilityImpl** (Time exerciseTime, Time length, Rate strike) const =0  
*implements the actual volatility calculation in derived classes*
- virtual [Volatility](#) **volatilityImpl** (const [Date](#) &exerciseDate, const [Period](#) &length, Rate strike) const
- void **checkRange** (Time, Time, Rate strike, bool extrapolate) const
- void **checkRange** (const [Date](#) &exerciseDate, const [Period](#) &length, Rate strike, bool extrapolate) const

## 7.748.2 Constructor & Destructor Documentation

### 7.748.2.1 [SwaptionVolatilityStructure \(\)](#)

default constructor

#### Warning

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

### 7.748.2.2 [SwaptionVolatilityStructure \(\)](#)

default constructor

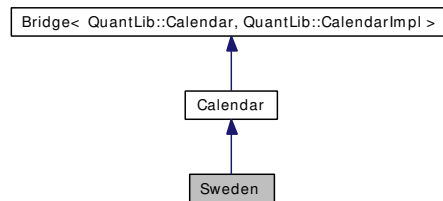
#### Warning

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.749 Sweden Class Reference

```
#include <ql/Calendars/sweden.hpp>
```

Inheritance diagram for Sweden:



### 7.749.1 Detailed Description

Swedish calendar.

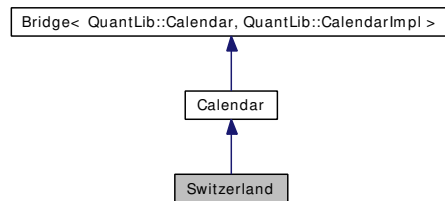
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- May Day, May 1st
- National Day, June 6th
- Midsummer Eve (Friday between June 18-24)
- Christmas Eve, December 24th
- Christmas Day, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31th

## 7.750 Switzerland Class Reference

```
#include <ql/Calendars/switzerland.hpp>
```

Inheritance diagram for Switzerland:



### 7.750.1 Detailed Description

Swiss 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

## 7.751 SymmetricSchurDecomposition Class Reference

```
#include <ql/Math/symmetricschurdecomposition.hpp>
```

### 7.751.1 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 transformations see "Matrix computation," second edition, by Golub and Van Loan, The Johns Hopkins University Press

#### Tests

the correctness of the returned values is tested by checking their properties.

### Public Member Functions

- [SymmetricSchurDecomposition](#) (const [Matrix](#) &s)
- const [Array](#) & **eigenvalues** () const
- const [Matrix](#) & **eigenvectors** () const

### 7.751.2 Constructor & Destructor Documentation

#### 7.751.2.1 [SymmetricSchurDecomposition](#) (const [Matrix](#) & s)

##### Precondition:

s must be symmetric

## 7.752 TabulatedGaussLegendre Class Reference

```
#include <ql/Math/gaussianquadratures.hpp>
```

### 7.752.1 Detailed Description

tabulated Gauss-Legendre quadratures

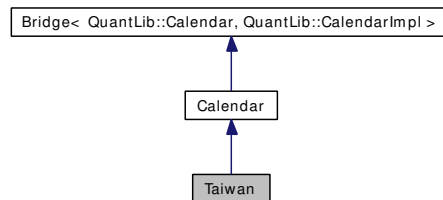
#### Public Member Functions

- **TabulatedGaussLegendre** (Size n=20)
- `template<class F> Real operator() (const F &f) const`
- `void order (Size)`
- `Size order () const`

## 7.753 Taiwan Class Reference

```
#include <ql/Calendars/taiwan.hpp>
```

Inheritance diagram for Taiwan:



### 7.753.1 Detailed Description

Taiwanese calendars.

Holidays for the [Taiwan](http://www.tse.com.tw/en/trading/trading_days.php) stock exchange (data from [http://www.tse.com.tw/en/trading/trading\\_days.php](http://www.tse.com.tw/en/trading/trading_days.php)):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Peace Memorial Day, February 28
- Labor Day, May 1st
- Double Tenth National Day, October 10th

Other holidays for which no rule is given (data available for 2002-2006 only:)

- Chinese Lunar New Year
- Tomb Sweeping Day
- Dragon Boat Festival
- Moon Festival

### Public Types

- enum [Market](#) { [TSEC](#) }

### Public Member Functions

- [Taiwan](#) ([Market](#) m=TSEC)



## 7.753.2 Member Enumeration Documentation

### 7.753.2.1 enum [Market](#)

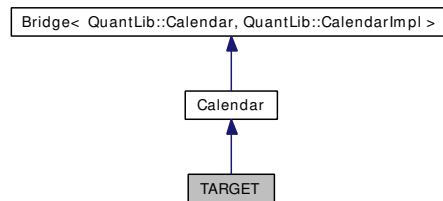
Enumerator:

*TSEC* [Taiwan](#) stock exchange.

## 7.754 TARGET Class Reference

```
#include <ql/Calendars/target.hpp>
```

Inheritance diagram for TARGET:



### 7.754.1 Detailed Description

TARGET calendar

Holidays (see <http://www.ecb.int>):

- 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)

#### Tests

the correctness of the returned results is tested against a list of known holidays.

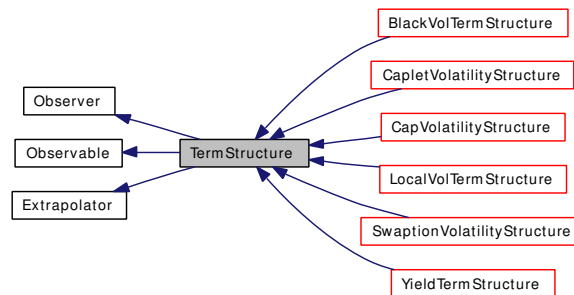
#### Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

## 7.755 TermStructure Class Reference

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

Inheritance diagram for TermStructure:



### 7.755.1 Detailed Description

Basic term-structure functionality.

### Public Member Functions

#### Dates

- virtual const [Date](#) & [referenceDate](#) () const  
*the date at which discount = 1.0 and/or variance = 0.0*
- virtual [Calendar](#) [calendar](#) () const  
*the calendar used for reference date calculation*
- virtual [Date](#) [maxDate](#) () const=0  
*the latest date for which the curve can return values*
- virtual Time [maxTime](#) () const  
*the latest time for which the curve can return values*
- virtual [DayCounter](#) [dayCounter](#) () const=0  
*the day counter used for date/time conversion*

#### Observer interface

- void [update](#) ()

### Protected Member Functions

- Time [timeFromReference](#) (const [Date](#) &date) const  
*date/time conversion*

- void [checkRange](#) (const [Date](#) &, bool extrapolate) const  
*date-range check*
- void [checkRange](#) (Time, bool extrapolate) const  
*time-range check*

## 7.755.2 Constructor & Destructor Documentation

### 7.755.2.1 [TermStructure](#) ()

default constructor

#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

### 7.755.2.2 [TermStructure](#) ()

default constructor

#### [Warning](#)

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.755.3 Member Function Documentation

### 7.755.3.1 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.

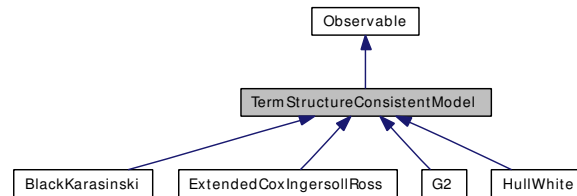
Implements [Observer](#).

Reimplemented in [ExtendedDiscountCurve](#), [FlatForward](#), [PiecewiseZeroSpreadedTermStructure](#), and [CapVolatilityVector](#).

## 7.756 TermStructureConsistentModel Class Reference

```
#include <ql/ShortRateModels/model.hpp>
```

Inheritance diagram for TermStructureConsistentModel:



### 7.756.1 Detailed Description

Term-structure consistent model class.

This is a base class for models that can reprice exactly any discount bond.

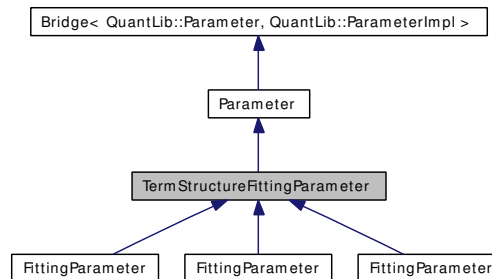
#### Public Member Functions

- `TermStructureConsistentModel` (const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- const [Handle](#)< [YieldTermStructure](#) > & `termStructure` () const

## 7.757 TermStructureFittingParameter Class Reference

#include <ql/ShortRateModels/parameter.hpp>

Inheritance diagram for TermStructureFittingParameter:



### 7.757.1 Detailed Description

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

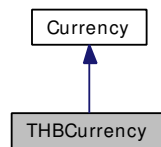
#### Public Member Functions

- `TermStructureFittingParameter` (const boost::shared\_ptr< Parameter::Impl > &impl)
- `TermStructureFittingParameter` (const [Handle](#)< [YieldTermStructure](#) > &term)

## 7.758 THBCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for THBCurrency:



### 7.758.1 Detailed Description

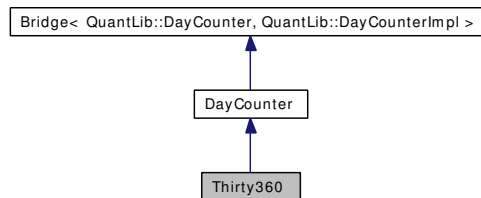
Thai baht.

The ISO three-letter code is THB; the numeric code is 764. It is divided in 100 stang.

## 7.759 Thirty360 Class Reference

```
#include <ql/DayCounters/thirty360.hpp>
```

Inheritance diagram for Thirty360:



### 7.759.1 Detailed Description

30/360 day count convention

The 30/360 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. Also known as "30/360", "360/360", or "Bond Basis"

European convention: starting dates or ending dates that occur on the 31st of a month become equal to the 30th of the same month. Also known as "30E/360", or "Eurobond Basis"

Italian convention: starting dates or ending dates that occur on February and are greater than 27 become equal to 30 for computational sake.

**Examples:**

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [Repo.cpp](#), and [swapvaluation.cpp](#).

### Public Types

- enum **Convention** {  
     **USA**, **BondBasis**, **European**, **EurobondBasis**,  
     **Italian** }

### Public Member Functions

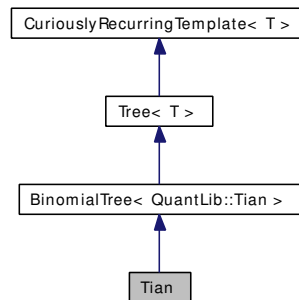
- **Thirty360** (Convention c=Thirty360::BondBasis)



## 7.760 Tian Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for Tian:



### 7.760.1 Detailed Description

Tian tree: third moment matching, multiplicative approach

#### Public Member Functions

- **Tian** (const boost::shared\_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)
- Real **underlying** (Size i, Size index) const
- Real **probability** (Size, Size, Size branch) const

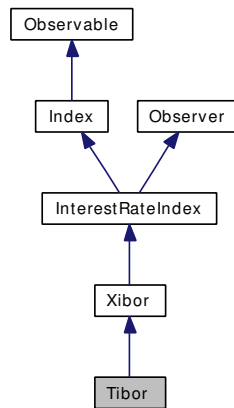
#### Protected Attributes

- Real **up\_**
- Real **down\_**
- Real **pu\_**
- Real **pd\_**

## 7.761 Tibor Class Reference

```
#include <ql/Indexes/tibor.hpp>
```

Inheritance diagram for Tibor:



### 7.761.1 Detailed Description

JPY TIBOR index

Tokyo Interbank Offered Rate.

#### Warning

This is the rate fixed in Tokio by JBA. Use [JPYLibor](#) if you're interested in the London fixing by BBA.

#### Todo

check settlement days.

### Public Member Functions

- **Tibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.762 TimeBasket Class Reference

```
#include <ql/CashFlows/timebasket.hpp>
```

### 7.762.1 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< Real > &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*

## 7.763 TimeGrid Class Reference

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

### 7.763.1 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.

#### Examples:

[BermudanSwaption.cpp](#).

#### sequence interface

- typedef std::vector< Time >::const\_iterator **const\_iterator**
- typedef std::vector< Time >::const\_reverse\_iterator **const\_reverse\_iterator**
- Time **operator[]** (Size i) const
- Time **at** (Size i) const
- Size **size** () const
- bool **empty** () const
- const\_iterator **begin** () const
- const\_iterator **end** () const
- const\_reverse\_iterator **rbegin** () const
- const\_reverse\_iterator **rend** () const
- Time **front** () const
- Time **back** () const

## Public Member Functions

#### Time grid interface

- Size [index](#) (Time t) const  
*returns the index i such that grid[i] = t*
- Size [closestIndex](#) (Time t) const  
*returns the index i such that grid[i] is closest to t*
- Time [closestTime](#) (Time t) const  
*returns the time on the grid closest to the given t*
- const std::vector< Time > & **mandatoryTimes** () const
- Time **dt** (Size i) const

## 7.763.2 Constructor & Destructor Documentation

### 7.763.2.1 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.

### 7.763.2.2 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.

### 7.763.2.3 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.

### 7.763.2.4 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.

## 7.764 TimeSeries Class Template Reference

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

### 7.764.1 Detailed Description

**template<class T, class Container = std::map<Date, T>> class QuantLib::TimeSeries< T, Container >**

Container for historical data.

This class acts as a generic repository for a set of historical data. Any single datum can be accessed through its date, while sets of consecutive data can be accessed through iterators.

#### Precondition:

The Container type must satisfy the requirements set by the C++ standard for associative containers.

#### Iterators

- typedef Container::const\_iterator **const\_iterator**
- typedef Container::const\_reverse\_iterator **const\_reverse\_iterator**
- const\_iterator **begin** () const
- const\_iterator **end** () const
- const\_reverse\_iterator **rbegin** () const
- const\_reverse\_iterator **rend** () const

#### Public Types

- typedef [Date](#) **key\_type**
- typedef T **value\_type**

#### Public Member Functions

- [TimeSeries](#) ()
- template<class DateIterator, class ValueIterator> [TimeSeries](#) (DateIterator dBegin, DateIterator dEnd, ValueIterator vBegin)
- template<class ValueIterator> [TimeSeries](#) (const [Date](#) &firstDate, ValueIterator begin, ValueIterator end)

#### Inspectors

- [Date firstDate](#) () const  
*returns the first date for which a historical datum exists*
- [Date lastDate](#) () const  
*returns the last date for which a historical datum exists*
- Size [size](#) () const

*returns the number of historical data including null ones*

- `bool empty () const`  
*returns whether the series contains any data*

#### Historical data access

- `T operator[] (const Date &d) const`  
*returns the (possibly null) datum corresponding to the given date*
- `T & operator[] (const Date &d)`

#### Utilities

- `const_iterator find (const Date &)`
- `std::vector< Date > dates () const`
- `std::vector< T > values () const`

## 7.764.2 Constructor & Destructor Documentation

### 7.764.2.1 TimeSeries ()

Default constructor

### 7.764.2.2 TimeSeries (DateIterator dBegin, DateIterator dEnd, ValueIterator vBegin)

This constructor initializes the history with a set of values passed as two sequences, the first containing dates and the second containing corresponding values.

### 7.764.2.3 TimeSeries (const Date &firstDate, ValueIterator begin, ValueIterator end)

This constructor initializes the history with a set of values. Such values are assigned to a corresponding number of consecutive dates starting from *firstDate* included.

## 7.765 TqrEigenDecomposition Class Reference

```
#include <ql/Math/tqreigendecomposition.hpp>
```

### 7.765.1 Detailed Description

tridiag. QR eigen decomposition with explicite shift aka Wilkinson

References:

Wilkinson, J.H. and Reinsch, C. 1971, [Linear](#) Algebra, vol. II of Handbook for Automatic Computation (New York: Springer-Verlag)

"Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery,

#### Tests

the correctness of the result is tested by checking it against known good values.

### Public Types

- enum **EigenVectorCalculation** { **WithEigenVector**, **WithoutEigenVector**, **OnlyFirstRowEigenVector** }
- enum **ShiftStrategy** { **NoShift**, **Overrelaxation**, **CloseEigenValue** }

### Public Member Functions

- **TqrEigenDecomposition** (const [Array](#) &diag, const [Array](#) &sub, EigenVectorCalculation calc=WithEigenVector, ShiftStrategy strategy=CloseEigenValue)
- const [Array](#) & **eigenvalues** () const
- const [Matrix](#) & **eigenvectors** () const
- Size **iterations** () const



## 7.766 TransformedGrid Class Reference

```
#include <ql/Math/transformedgrid.hpp>
```

### 7.766.1 Detailed Description

transformed grid

This package encapsulates an array of grid points. It is used primarily in PDE calculations.

### Public Member Functions

- **TransformedGrid** (const [Array](#) &grid)
- template<class T> **TransformedGrid** (const [Array](#) &grid, T func)
- const [Array](#) & **gridArray** () const
- const [Array](#) & **transformedGridArray** () const
- const [Array](#) & **dxmArray** () const
- const [Array](#) & **dxpArray** () const
- const [Array](#) & **dxArray** () const
- [Real](#) **grid** (Size i) const
- [Real](#) **transformedGrid** (Size i) const
- [Real](#) **dxm** (Size i) const
- [Real](#) **dxp** (Size i) const
- [Real](#) **dx** (Size i) const
- Size **size** () const

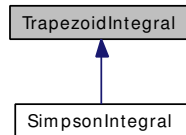
### Protected Attributes

- [Array](#) **grid\_**
- [Array](#) **transformedGrid\_**
- [Array](#) **dxm\_**
- [Array](#) **dxp\_**
- [Array](#) **dx\_**

## 7.767 TrapezoidIntegral Class Reference

```
#include <ql/Math/trapezoidintegral.hpp>
```

Inheritance diagram for TrapezoidIntegral:



### 7.767.1 Detailed Description

Integral of a one-dimensional function.

Given a target accuracy  $\epsilon$ , the integral of a function  $f$  between  $a$  and  $b$  is calculated by means of the trapezoid formula

$$\int_a^b f dx = \frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \dots + f(x_{N-1}) + \frac{1}{2}f(x_N)$$

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.

#### Tests

the correctness of the result is tested by checking it against known good values.

### Public Types

- enum **Method** { **Default**, **MidPoint** }

### Public Member Functions

- **TrapezoidIntegral** (Real accuracy, Method method=Default, Size maxIterations=Null< Size >())
- template<class F> Real **operator()** (const F &f, Real a, Real b) const
- Real **accuracy** () const
- Real & **accuracy** ()
- Method **method** () const
- Method & **method** ()
- Size **maxIterations** () const
- Size & **maxIterations** ()

### Protected Member Functions

- template<class F> Real **defaultIteration** (const F &f, Real a, Real b, Real I, Size N) const
- template<class F> Real **midPointIteration** (const F &f, Real a, Real b, Real I, Size N) const

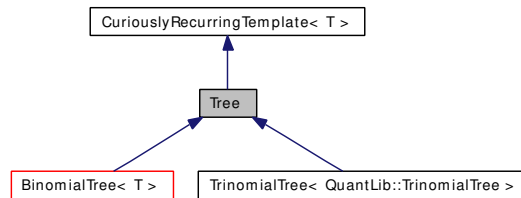
## Protected Attributes

- Real `accuracy_`
- Method `method_`
- Size `maxIterations_`

## 7.768 Tree Class Template Reference

```
#include <ql/Lattices/tree.hpp>
```

Inheritance diagram for Tree:



### 7.768.1 Detailed Description

```
template<class T> class QuantLib::Tree< T >
```

Tree approximating a single-factor diffusion

Derived classes must implement the following interface:

```
public:
    Real underlying(Size i, Size index) const;
    Size size(Size i) const;
    Size descendant(Size i, Size index, Size branch) const;
    Real probability(Size i, Size index, Size branch) const;
```

and provide a public enumeration

```
enum { branches = N };
```

where N is a suitable constant (2 for binomial, 3 for trinomial...)

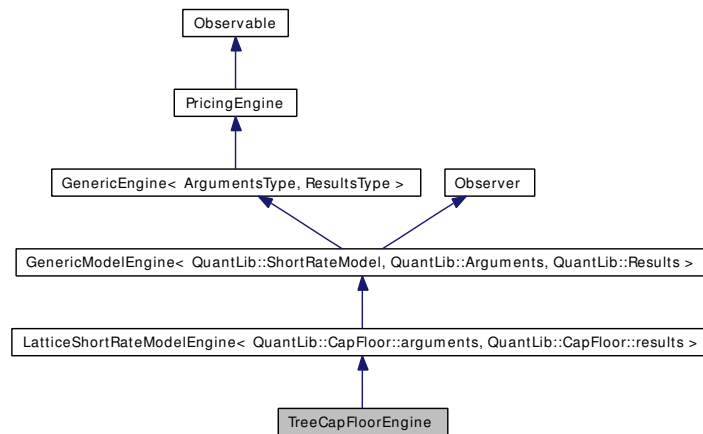
### Public Member Functions

- **Tree** (Size columns)
- Size **columns** () const

## 7.769 TreeCapFloorEngine Class Reference

#include <ql/PricingEngines/CapFloor/treecapfloorengine.hpp>

Inheritance diagram for TreeCapFloorEngine:



### 7.769.1 Detailed Description

Numerical lattice engine for cap/floors.

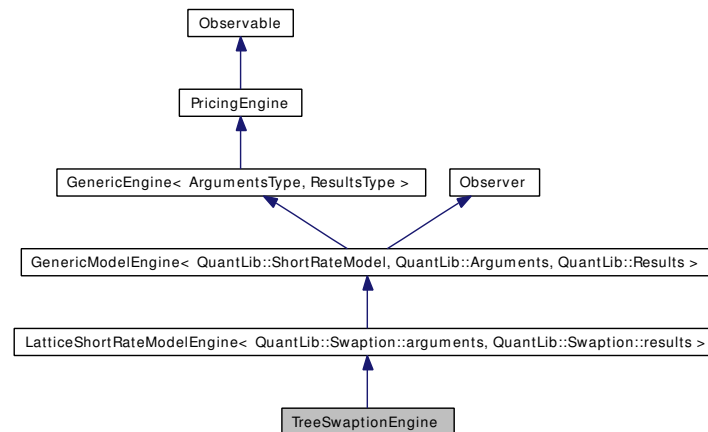
#### Public Member Functions

- **TreeCapFloorEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &model, Size timeSteps)
- **TreeCapFloorEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &model, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

## 7.770 TreeSwaptionEngine Class Reference

#include <ql/PricingEngines/Swaption/treeswaptionengine.hpp>

Inheritance diagram for TreeSwaptionEngine:



### 7.770.1 Detailed Description

Numerical lattice engine for swaptions.

#### Warning

This engine is not guaranteed to work if the underlying swap has a start date in the past, i.e., before today's date. When using this engine, prune the initial part of the swap so that it starts at  $t \geq 0$ .

#### Tests

calculations are checked against cached results

#### Examples:

[BermudanSwaption.cpp](#).

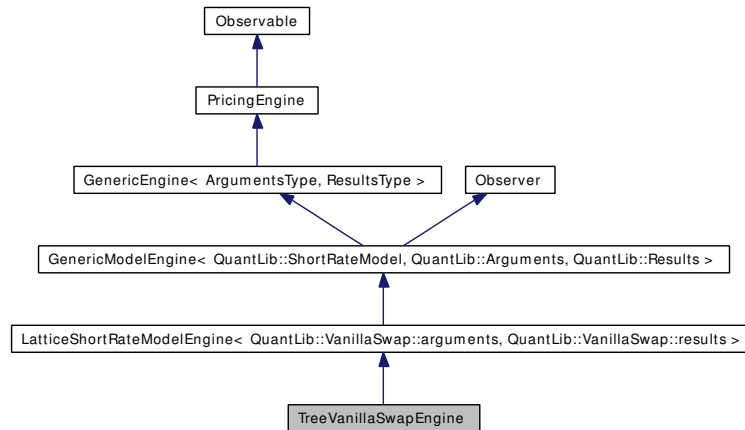
### Public Member Functions

- **TreeSwaptionEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &, Size timeSteps)
- **TreeSwaptionEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

## 7.771 TreeVanillaSwapEngine Class Reference

#include <ql/PricingEngines/Swaption/treeswaptionengine.hpp>

Inheritance diagram for TreeVanillaSwapEngine:



### 7.771.1 Detailed Description

Numerical lattice engine for simple swaps.

#### Tests

calculations are checked against known good results

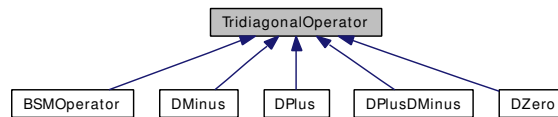
### Public Member Functions

- **TreeVanillaSwapEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &, Size timeSteps)
- **TreeVanillaSwapEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

## 7.772 TridiagonalOperator Class Reference

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Inheritance diagram for TridiagonalOperator:



### 7.772.1 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, Real tol) const`  
*solve linear system with SOR approach*
- static `Disposable< TridiagonalOperator > identity (Size size)`  
*identity instance*

### Public Types

- typedef `Array array_type`

### 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** (Real, Real)
- void **setMidRow** (Size, Real, Real, Real)
- void **setMidRows** (Real, Real, Real)
- void **setLastRow** (Real, Real)
- void **setTime** (Time t)

### Utilities

- void **swap** ([TridiagonalOperator](#) &)

### Protected Attributes

- [Array](#) **diagonal\_**
- [Array](#) **lowerDiagonal\_**
- [Array](#) **upperDiagonal\_**
- boost::shared\_ptr< [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** \* (Real, const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator** \* (const [TridiagonalOperator](#) &, Real)
- [Disposable](#)< [TridiagonalOperator](#) > **operator** / (const [TridiagonalOperator](#) &, Real)

### Classes

- class [TimeSetter](#)  
*encapsulation of time-setting logic*

## 7.773 TridiagonalOperator::TimeSetter Class Reference

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

### 7.773.1 Detailed Description

encapsulation of time-setting logic

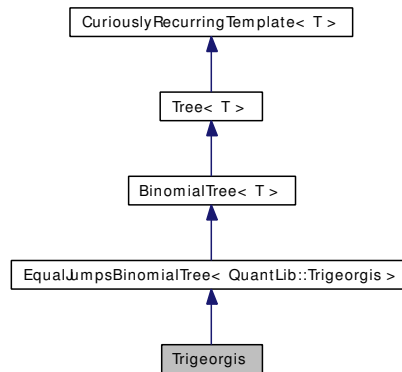
### Public Member Functions

- virtual void **setTime** (Time t, [TridiagonalOperator](#) &L) const=0

## 7.774 Trigeorgis Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for Trigeorgis:



### 7.774.1 Detailed Description

Trigeorgis (additive equal jumps) binomial tree

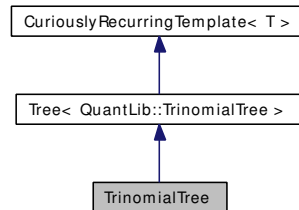
#### Public Member Functions

- **Trigeorgis** (const boost::shared\_ptr< [StochasticProcess1D](#) > &, Time end, Size steps, Real strike)

## 7.775 TrinomialTree Class Reference

```
#include <ql/Lattices/trinomialtree.hpp>
```

Inheritance diagram for TrinomialTree:



### 7.775.1 Detailed Description

Recombining trinomial tree class.

This class defines a recombining trinomial tree approximating a 1-D stochastic process.

#### Warning

The diffusion term of the SDE must be independent of the underlying process.

### Public Types

- enum { **branches** = 3 }

### Public Member Functions

- **TrinomialTree** (const boost::shared\_ptr< [StochasticProcess1D](#) > &process, const [TimeGrid](#) &timeGrid, bool isPositive=false)
- Real **dx** (Size i) const
- const [TimeGrid](#) & **timeGrid** () const
- Size **size** (Size i) const
- Real **underlying** (Size i, Size index) const
- Size **descendant** (Size i, Size index, Size branch) const
- Real **probability** (Size i, Size index, Size branch) const

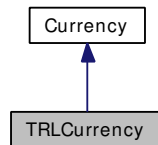
### Protected Attributes

- std::vector< Branching > **branchings\_**
- Real **x0\_**
- std::vector< Real > **dx\_**
- [TimeGrid](#) **timeGrid\_**

## 7.776 TRLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for TRLCurrency:



### 7.776.1 Detailed Description

Turkish lira.

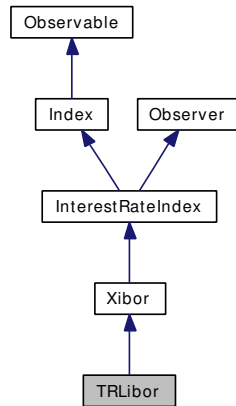
The ISO three-letter code was TRL; the numeric code was 792. It was divided in 100 kurus.

Obsoleted by the new Turkish lira since 2005.

## 7.777 TRLibor Class Reference

```
#include <ql/Indexes/trlibor.hpp>
```

Inheritance diagram for TRLibor:



### 7.777.1 Detailed Description

TRY LIBOR rate

TRY LIBOR fixed by TBA.

See <<http://www.trlibor.org/trlibor/english/default.asp>>

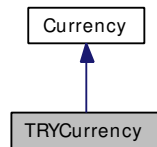
### Public Member Functions

- **TRLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## 7.778 TRYCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for TRYCurrency:



### 7.778.1 Detailed Description

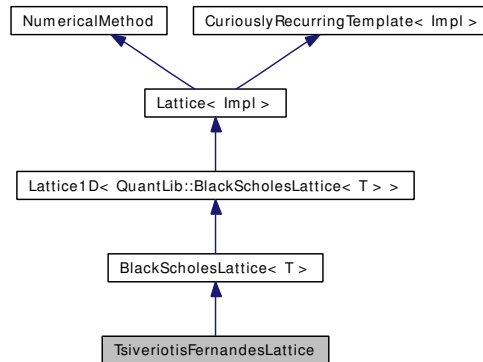
New Turkish lira.

The ISO three-letter code is TRY; the numeric code is 949. It is divided in 100 new kurus.

## 7.779 TsiveriotisFernandesLattice Class Template Reference

```
#include <ql/Lattices/tflattice.hpp>
```

Inheritance diagram for TsiveriotisFernandesLattice:



### 7.779.1 Detailed Description

```
template<class T> class QuantLib::TsiveriotisFernandesLattice< T >
```

Binomial lattice approximating the Tsiveriotis-Fernandes model.

### Public Member Functions

- **TsiveriotisFernandesLattice** (const boost::shared\_ptr< T > &tree, Rate riskFreeRate, Time end, Size steps, Real creditSpread, Volatility volatility, Spread divYield)
- Rate **riskFreeRate** () const
- Real **creditSpread** () const
- Real **dt** () const

### Protected Member Functions

- void **stepback** (Size i, const [Array](#) &values, const [Array](#) &conversionProbability, const [Array](#) &spreadAdjustedRate, [Array](#) &newValues, [Array](#) &newConversionProbability, [Array](#) &newSpreadAdjustedRate) const
- void **rollback** ([DiscretizedAsset](#) &, Time to) const
- void **partialRollback** ([DiscretizedAsset](#) &, Time to) const

### 7.779.2 Member Function Documentation

**7.779.2.1 void rollback** ([DiscretizedAsset](#) &, Time to) const [protected, virtual]

Roll back an asset until the given time, performing any needed adjustment.

Reimplemented from [Lattice](#).



**7.779.2.2 void partialRollback ([DiscretizedAsset](#) &, Time *to*) const** [protected, virtual]

Roll back an asset until the given time, but do not perform the final adjustment.

**Warning**

In version 0.3.7 and earlier, this method was called `rollAlmostBack` method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

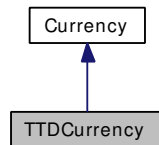
```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Reimplemented from [Lattice](#).

## 7.780 TTDCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for TTDCurrency:



### 7.780.1 Detailed Description

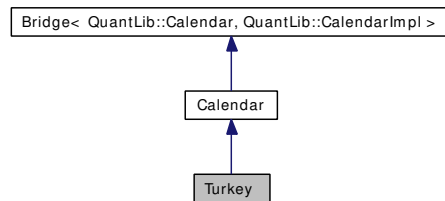
Trinidad & Tobago dollar.

The ISO three-letter code is TTD; the numeric code is 780. It is divided in 100 cents.

## 7.781 Turkey Class Reference

```
#include <ql/Calendars/turkey.hpp>
```

Inheritance diagram for Turkey:



### 7.781.1 Detailed Description

Turkish calendar.

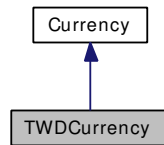
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- National Holidays (April 23rd, May 19th, August 30th, October 29th)
- Local Holidays (Kurban, Ramadan; 2004 to 2009 only)

## 7.782 TWDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for TWDCurrency:



### 7.782.1 Detailed Description

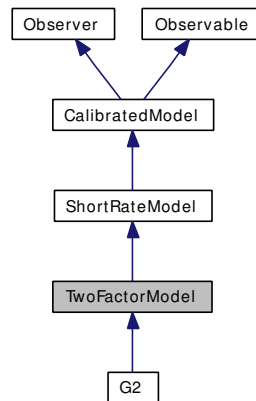
[Taiwan](#) dollar.

The ISO three-letter code is TWD; the numeric code is 901. It is divided in 100 cents.

## 7.783 TwoFactorModel Class Reference

```
#include <ql/ShortRateModels/twofactormodel.hpp>
```

Inheritance diagram for TwoFactorModel:



### 7.783.1 Detailed Description

Abstract base-class for two-factor models.

#### Public Member Functions

- **TwoFactorModel** (Size nParams)
- virtual boost::shared\_ptr< [ShortRateDynamics](#) > [dynamics](#) () const=0  
*Returns the short-rate dynamics.*
- boost::shared\_ptr< [NumericalMethod](#) > [tree](#) (const [TimeGrid](#) &grid) const  
*Returns a two-dimensional trinomial tree.*

#### Classes

- class [ShortRateDynamics](#)  
*Class describing the dynamics of the two state variables.*
- class [ShortRateTree](#)  
*Recombining two-dimensional tree discretizing the state variable.*

## 7.784 TwoFactorModel::ShortRateDynamics Class Reference

```
#include <ql/ShortRateModels/twofactormodel.hpp>
```

### 7.784.1 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

$$dW_t^x dW_t^y = \rho dt$$

.

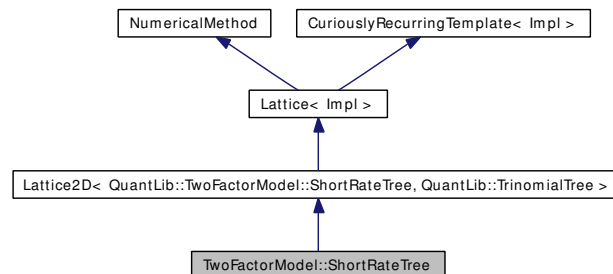
### Public Member Functions

- **ShortRateDynamics** (const boost::shared\_ptr< [StochasticProcess1D](#) > &xProcess, const boost::shared\_ptr< [StochasticProcess1D](#) > &yProcess, Real correlation)
- virtual [Rate](#) **shortRate** (Time t, Real x, Real y) const=0
- const boost::shared\_ptr< [StochasticProcess1D](#) > & **xProcess** () const  
*Risk-neutral dynamics of the first state variable x.*
- const boost::shared\_ptr< [StochasticProcess1D](#) > & **yProcess** () const  
*Risk-neutral dynamics of the second state variable y.*
- Real **correlation** () const  
*Correlation  $\rho$  between the two brownian motions.*
- boost::shared\_ptr< [StochasticProcess](#) > **process** () const  
*Joint process of the two variables.*

## 7.785 TwoFactorModel::ShortRateTree Class Reference

```
#include <ql/ShortRateModels/twofactormodel.hpp>
```

Inheritance diagram for TwoFactorModel::ShortRateTree:



### 7.785.1 Detailed Description

Recombining two-dimensional tree discretizing the state variable.

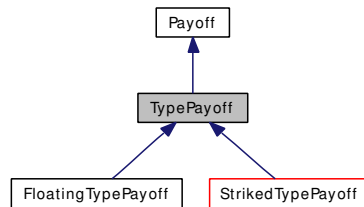
#### Public Member Functions

- [ShortRateTree](#) (const boost::shared\_ptr< [TrinomialTree](#) > &tree1, const boost::shared\_ptr< [TrinomialTree](#) > &tree2, const boost::shared\_ptr< [ShortRateDynamics](#) > &dynamics)  
*Plain tree build-up from short-rate dynamics.*
- [DiscountFactor](#) **discount** (Size i, Size index) const

## 7.786 TypePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for TypePayoff:



### 7.786.1 Detailed Description

Intermediate class for call/put/straddle payoffs.

#### Public Member Functions

- **TypePayoff** (Option::Type type)
- Option::Type **optionType** () const

#### Protected Attributes

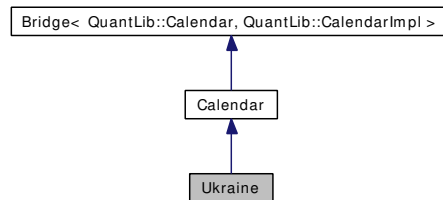
- Option::Type **type\_**



## 7.787 Ukraine Class Reference

```
#include <ql/Calendars/ukraine.hpp>
```

Inheritance diagram for Ukraine:



### 7.787.1 Detailed Description

Ukrainian calendars.

Holidays for the Ukrainian stock exchange (data from <http://www.ukrse.kiev.ua/eng/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Orthodox Christmas, January 7th
- International Women's Day, March 8th
- Easter Monday
- Holy Trinity Day, 50 days after Easter
- International Workers' Solidarity Days, May 1st and 2nd
- Victory Day, May 9th
- Constitution Day, June 28th
- Independence Day, August 24th

Holidays falling on a Saturday or Sunday are moved to the following Monday.

### Public Types

- enum [Market](#) { [USE](#) }

### Public Member Functions

- [Ukraine](#) ([Market](#) m=USE)

## 7.787.2 Member Enumeration Documentation

### 7.787.2.1 enum [Market](#)

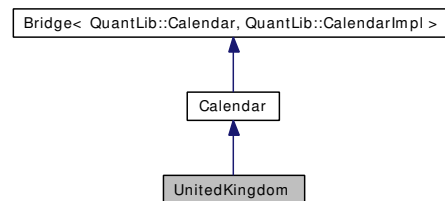
Enumerator:

*USE* Ukrainian stock exchange.

## 7.788 UnitedKingdom Class Reference

```
#include <ql/Calendars/unitedkingdom.hpp>
```

Inheritance diagram for UnitedKingdom:



### 7.788.1 Detailed Description

United Kingdom calendars.

Public holidays (data from <http://www.dti.gov.uk/er/bankhol.htm>):

- 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 Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Holidays for the stock exchange:

- 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 Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Holidays for the metals exchange:

- 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 Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

### Todo

add LIFFE

### Tests

the correctness of the returned results is tested against a list of known holidays.

## Public Types

- enum [Market](#) { [Settlement](#), [Exchange](#), [Metals](#) }  
*UK calendars.*

## Public Member Functions

- [UnitedKingdom](#) ([Market](#) market=[Settlement](#))

## 7.788.2 Member Enumeration Documentation

### 7.788.2.1 enum [Market](#)

UK calendars.

**Enumerator:**

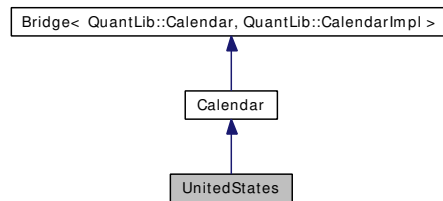
*Settlement* generic settlement calendar

*Exchange* London stock-exchange calendar.

## 7.789 UnitedStates Class Reference

```
#include <ql/Calendars/unitedstates.hpp>
```

Inheritance diagram for UnitedStates:



### 7.789.1 Detailed Description

United States calendars.

Public holidays (see: <http://www.opm.gov/fedhol/>):

- 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
- Presidents' Day (a.k.a. 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
- Veterans' 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)

Holidays for the stock exchange (data from <http://www.nyse.com>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Martin Luther King's birthday, third Monday in January (since 1998)
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Good Friday

- 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
- Thanksgiving Day, fourth Thursday in November
- Presidential election day, first Tuesday in November of election years (until 1980)
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)
- Special historic closings (see <http://www.nyse.com/about/1022221392381.html> )

Holidays for the government bond market (data from <http://www.bondmarkets.com>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Martin Luther King's birthday, third Monday in January
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Good Friday
- 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
- Veterans' 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)

Holidays for the North American Energy Reliability Council (data from <http://www.nerc.com/~oc/offpeaks.html>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday)
- Labor Day, first Monday in September
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday)

## Tests

the correctness of the returned results is tested against a list of known holidays.

## Public Types

- enum [Market](#) { [Settlement](#), [NYSE](#), [GovernmentBond](#), [NERC](#) }  
*US calendars.*

## Public Member Functions

- [UnitedStates](#) ([Market](#) market=[Settlement](#))

## 7.789.2 Member Enumeration Documentation

### 7.789.2.1 enum [Market](#)

US calendars.

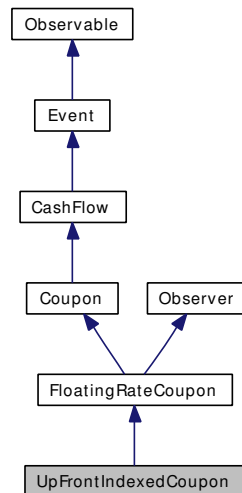
#### Enumerator:

*Settlement* generic settlement calendar  
*NYSE* New York stock exchange calendar.  
*GovernmentBond* government-bond calendar  
*NERC* off-peak days for NERC

## 7.790 UpFrontIndexedCoupon Class Reference

```
#include <ql/CashFlows/upfrontindexedcoupon.hpp>
```

Inheritance diagram for UpFrontIndexedCoupon:



### 7.790.1 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** (const [Date](#) &paymentDate, const [Real](#) nominal, const [Date](#) &startDate, const [Date](#) &endDate, const [Integer](#) fixingDays, const boost::shared\_ptr<[InterestRateIndex](#) > &index, const [Real](#) gearing=1.0, const [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())

#### Visitability

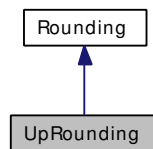
- virtual void **accept** ([AcyclicVisitor](#) &)



## 7.791 UpRounding Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for UpRounding:



### 7.791.1 Detailed Description

Up-rounding.

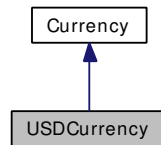
#### Public Member Functions

- `UpRounding` ([Integer](#) precision, [Integer](#) digit=5)

## 7.792 USDCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for USDCurrency:



### 7.792.1 Detailed Description

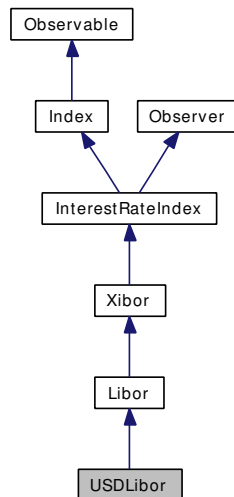
U.S. dollar.

The ISO three-letter code is USD; the numeric code is 840. It is divided in 100 cents.

## 7.793 USDLibor Class Reference

```
#include <ql/Indexes/usdlibor.hpp>
```

Inheritance diagram for USDLibor:



### 7.793.1 Detailed Description

USD LIBOR rate

US Dollar LIBOR fixed by BBA.

See <<http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1414>>.

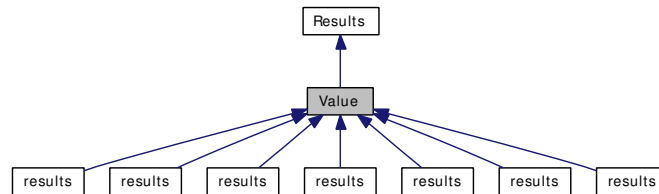
### Public Member Functions

- **USDLibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >(), [BusinessDayConvention](#) convention=MonthEndReference, [Integer](#) settlementDays=2)

## 7.794 Value Class Reference

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

Inheritance diagram for Value:



### 7.794.1 Detailed Description

pricing results

#### Public Member Functions

- `void reset ()`

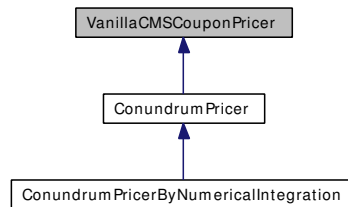
#### Public Attributes

- Real `value`
- Real `errorEstimate`

## 7.795 VanillaCMSCouponPricer Class Reference

```
#include <ql/CashFlows/cmscoupon.hpp>
```

Inheritance diagram for VanillaCMSCouponPricer:



### 7.795.1 Detailed Description

pricer for vanilla CMS coupons

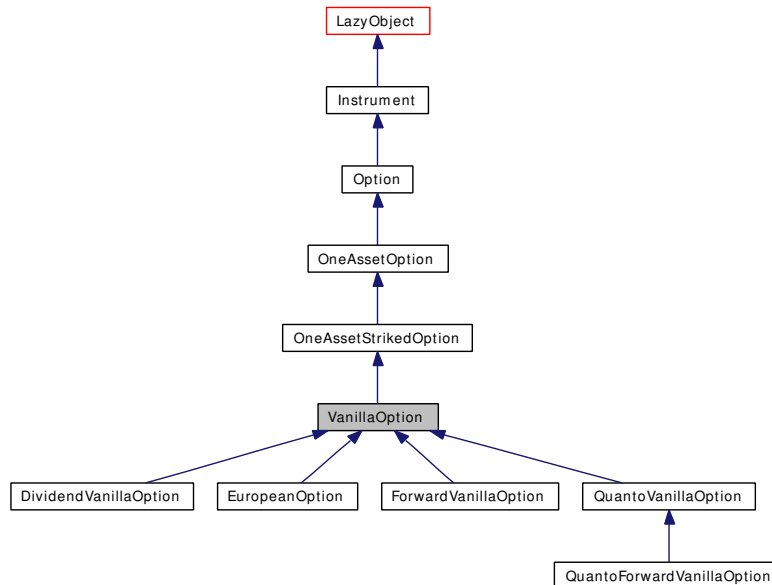
#### Public Member Functions

- virtual [Real](#) **price** () const=0
- virtual [Real](#) **rate** () const=0
- virtual void **initialize** (const [CMSCoupon](#) &coupon)=0

## 7.796 VanillaOption Class Reference

```
#include <ql/Instruments/vanillaoption.hpp>
```

Inheritance diagram for VanillaOption:



### 7.796.1 Detailed Description

Vanilla option (no discrete dividends, no barriers) on a single asset.

**Examples:**

[EquityOption.cpp](#).

### Public Member Functions

- **VanillaOption** (const boost::shared\_ptr< [StochasticProcess](#) > &, const boost::shared\_ptr< [StrikedTypePayoff](#) > &, const boost::shared\_ptr< [Exercise](#) > &, const boost::shared\_ptr< [PricingEngine](#) > &engine=boost::shared\_ptr< [PricingEngine](#) >())

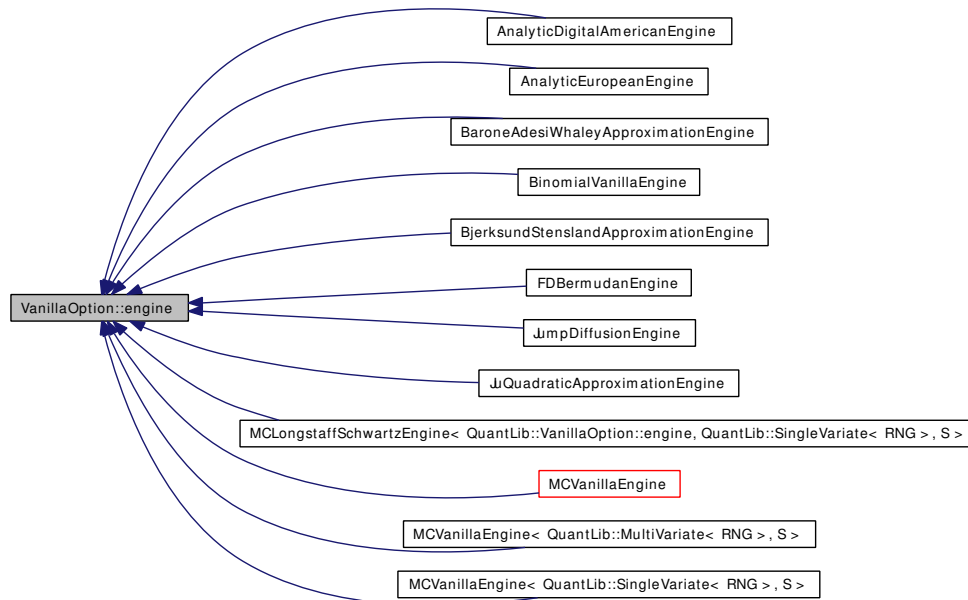
### Classes

- class [engine](#)  
*Vanilla option engine base class.*

## 7.797 VanillaOption::engine Class Reference

```
#include <ql/Instruments/vanillaoption.hpp>
```

Inheritance diagram for VanillaOption::engine:



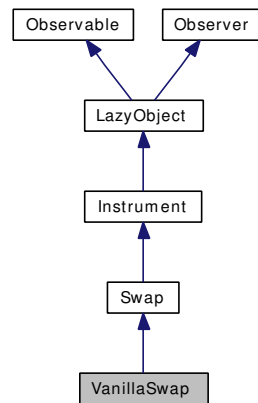
### 7.797.1 Detailed Description

Vanilla option engine base class.

## 7.798 VanillaSwap Class Reference

```
#include <ql/Instruments/vanillaswap.hpp>
```

Inheritance diagram for VanillaSwap:



### 7.798.1 Detailed Description

Plain-vanilla swap.

#### Tests

- the correctness of the returned value is tested by checking that the price of a swap paying the fair fixed rate is null.
- the correctness of the returned value is tested by checking that the price of a swap receiving the fair floating-rate spread is null.
- the correctness of the returned value is tested by checking that the price of a swap decreases with the paid fixed rate.
- the correctness of the returned value is tested by checking that the price of a swap increases with the received floating-rate spread.
- the correctness of the returned value is tested by checking it against a known good value.

#### Examples:

[BermudanSwaption.cpp](#), and [swapvaluation.cpp](#).

### Public Member Functions

- **VanillaSwap** (bool payFixedRate, Real nominal, const [Schedule](#) &fixedSchedule, Rate fixedRate, const [DayCounter](#) &fixedDayCount, const [Schedule](#) &floatSchedule, const boost::shared\_ptr< [Xibor](#) > &index, [Integer](#) indexFixingDays, Spread spread, const [DayCounter](#) &floatingDayCount, const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- **VanillaSwap** (bool payFixedRate, Real nominal, const [Schedule](#) &fixedSchedule, Rate fixedRate, const [DayCounter](#) &fixedDayCount, const [Schedule](#) &floatSchedule, const boost::shared\_ptr< [Xibor](#) > &index, Spread spread, const [DayCounter](#) &floatingDayCount, const [Handle](#)< [YieldTermStructure](#) > &termStructure)



- Rate **fairRate** () const
- Spread **fairSpread** () const
- Real **fixedLegBPS** () const
- Real **floatingLegBPS** () const
- Rate **fixedRate** () const
- Spread **spread** () const
- Real **nominal** () const
- bool **payFixedRate** () const
- const std::vector< boost::shared\_ptr< [CashFlow](#) > > & **fixedLeg** () const
- const std::vector< boost::shared\_ptr< [CashFlow](#) > > & **floatingLeg** () const
- void **setupArguments** ([Arguments](#) \*args) const
- void **fetchResults** (const [Results](#) \*) const

## Classes

- class [arguments](#)  
*Arguments for simple swap calculation*
- class [results](#)  
*Results from simple swap calculation*

## 7.798.2 Constructor & Destructor Documentation

- 7.798.2.1 **VanillaSwap** (bool *payFixedRate*, Real *nominal*, const [Schedule](#) & *fixedSchedule*, Rate *fixedRate*, const [DayCounter](#) & *fixedDayCount*, const [Schedule](#) & *floatSchedule*, const boost::shared\_ptr< [Xibor](#) > & *index*, [Integer](#) *indexFixingDays*, Spread *spread*, const [DayCounter](#) & *floatingDayCount*, const [Handle](#)< [YieldTermStructure](#) > & *termStructure*)

### Deprecated

use the other [VanillaSwap](#) constructor or [Swap](#) instead

## 7.798.3 Member Function Documentation

- 7.798.3.1 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](#).

- 7.798.3.2 void **fetchResults** (const [Results](#) \*) const [virtual]

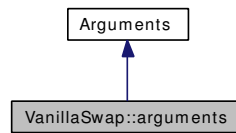
When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

## 7.799 VanillaSwap::arguments Class Reference

```
#include <ql/Instruments/vanillaswap.hpp>
```

Inheritance diagram for VanillaSwap::arguments:



### 7.799.1 Detailed Description

Arguments for simple swap calculation

#### Public Member Functions

- void **validate** () const

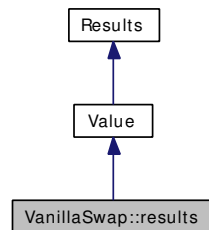
#### Public Attributes

- bool **payFixed**
- Real **nominal**
- std::vector< Time > **fixedResetTimes**
- std::vector< Time > **fixedPayTimes**
- std::vector< Real > **fixedCoupons**
- std::vector< Time > **floatingAccrualTimes**
- std::vector< Time > **floatingResetTimes**
- std::vector< Time > **floatingFixingTimes**
- std::vector< Time > **floatingPayTimes**
- std::vector< Spread > **floatingSpreads**
- Real **currentFloatingCoupon**

## 7.800 VanillaSwap::results Class Reference

```
#include <ql/Instruments/vanillaswap.hpp>
```

Inheritance diagram for VanillaSwap::results:



### 7.800.1 Detailed Description

Results from simple swap calculation

#### Public Member Functions

- void **reset** ()

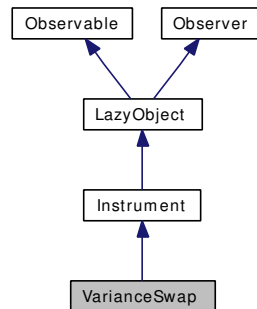
#### Public Attributes

- Real **fixedLegBPS**
- Real **floatingLegBPS**
- Rate **fairRate**
- Spread **fairSpread**

## 7.801 VarianceSwap Class Reference

```
#include <ql/Instruments/varianceswap.hpp>
```

Inheritance diagram for VarianceSwap:



### 7.801.1 Detailed Description

Variance swap.

#### Warning

This class does not manage seasoned variance swaps.

### Public Types

- typedef std::vector< std::pair< boost::shared\_ptr< [StrikedTypePayoff](#) >, Real > > **WeightsType**

### Public Member Functions

- **VarianceSwap** (Position::Type position, Real strike, Real notional, const boost::shared\_ptr< [StochasticProcess](#) > &process, const [Date](#) &maturityDate, const boost::shared\_ptr< [PricingEngine](#) > &engine)
- void [setupArguments](#) ([Arguments](#) \*args) const
- void [fetchResults](#) (const [Results](#) \*) const

#### Instrument interface

- bool [isExpired](#) () const  
*returns whether the instrument is still tradable.*

#### Additional interface

- Real **strike** () const
- Position::Type **position** () const
- [Date](#) **maturityDate** () const
- [Date](#) **settlementDate** () const
- Real **notional** () const
- Real **fairVariance** () const
- std::vector< std::pair< Real, Real > > **optionWeights** (Option::Type) const

## Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

## Protected Attributes

- boost::shared\_ptr< [GeneralizedBlackScholesProcess](#) > **process\_**
- Position::Type **position\_**
- Real **strike\_**
- Real **notional\_**
- [Date](#) **maturityDate\_**
- WeightsType **optionWeights\_**
- Real **fairVariance\_**

## Classes

- class [arguments](#)  
*Arguments for forward fair-variance calculation*
- class [engine](#)  
*base class for variance-swap engines*
- class [results](#)  
*Results from variance-swap calculation*

## 7.801.2 Member Function Documentation

### 7.801.2.1 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](#).

### 7.801.2.2 void [fetchResults](#) (const [Results](#) \*) const [virtual]

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

Reimplemented from [Instrument](#).

### 7.801.2.3 void [setupExpired](#) () const [protected, virtual]

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

Reimplemented from [Instrument](#).

**7.801.2.4 void performCalculations () const** [protected, virtual]

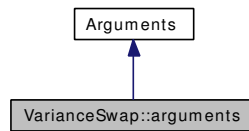
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

## 7.802 VarianceSwap::arguments Class Reference

```
#include <ql/Instruments/varianceswap.hpp>
```

Inheritance diagram for VarianceSwap::arguments:



### 7.802.1 Detailed Description

Arguments for forward fair-variance calculation

#### Public Member Functions

- void **validate** () const

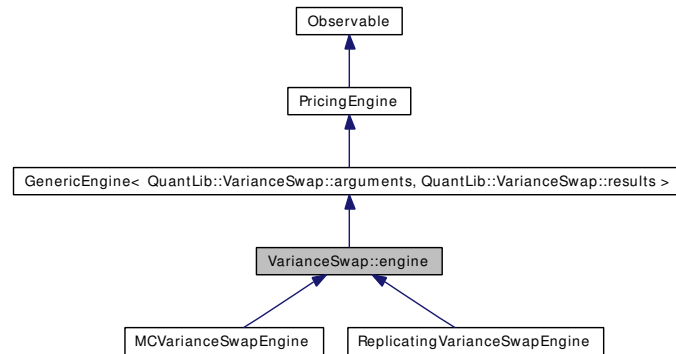
#### Public Attributes

- boost::shared\_ptr< [GeneralizedBlackScholesProcess](#) > **stochasticProcess**
- Position::Type **position**
- Real **strike**
- Real **notional**
- [Date](#) **maturityDate**

## 7.803 VarianceSwap::engine Class Reference

```
#include <ql/Instruments/varianceswap.hpp>
```

Inheritance diagram for VarianceSwap::engine:



### 7.803.1 Detailed Description

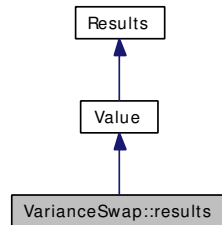
base class for variance-swap engines



## 7.804 VarianceSwap::results Class Reference

```
#include <ql/Instruments/varianceswap.hpp>
```

Inheritance diagram for VarianceSwap::results:



### 7.804.1 Detailed Description

Results from variance-swap calculation

#### Public Member Functions

- void **reset** ()

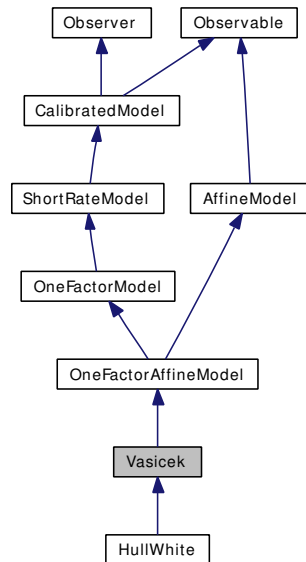
#### Public Attributes

- Real **fairVariance**
- WeightsType **optionWeights**
- WeightsType::const\_iterator **iterator**

## 7.805 Vasicek Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>
```

Inheritance diagram for Vasicek:



### 7.805.1 Detailed Description

Vasicek model class

This class implements the [Vasicek](#) model defined by

$$dr_t = a(b - r_t)dt + \sigma dW_t,$$

where  $a$ ,  $b$  and  $\sigma$  are constants; a risk premium  $\lambda$  can also be specified.

#### Public Member Functions

- **Vasicek** ([Rate](#) r0=0.05, Real a=0.1, Real b=0.05, Real sigma=0.01, Real lambda=0.0)
- virtual Real **discountBondOption** (Option::Type type, Real strike, Time maturity, Time bondMaturity) const
- virtual boost::shared\_ptr< ShortRateDynamics > **dynamics** () const  
*returns the short-rate dynamics*

#### Protected Member Functions

- virtual Real **A** (Time t, Time T) const
- virtual Real **B** (Time t, Time T) const
- Real **a** () const
- Real **b** () const
- Real **lambda** () const
- Real **sigma** () const

## Protected Attributes

- Real `r0_`
- [Parameter](#) & `a_`
- [Parameter](#) & `b_`
- [Parameter](#) & `sigma_`
- [Parameter](#) & `lambda_`

## Classes

- class [Dynamics](#)  
*Short-rate dynamics in the Vasicek model.*

## 7.806 Vasicek::Dynamics Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>
```

### 7.806.1 Detailed Description

Short-rate dynamics in the Vasicek model.

The short-rate follows an Ornstein-Uhlenbeck process with mean  $b$ .

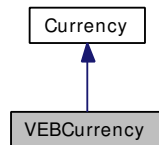
### Public Member Functions

- **Dynamics** (Real  $a$ , Real  $b$ , Real  $\sigma$ , Real  $r_0$ )
- virtual Real **variable** ([Time](#), [Rate](#)  $r$ ) const
- virtual Real **shortRate** ([Time](#), Real  $x$ ) const

## 7.807 VEBCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for VEBCurrency:



### 7.807.1 Detailed Description

Venezuelan bolivar.

The ISO three-letter code is VEB; the numeric code is 862. It is divided in 100 centimos.

## 7.808 Visitor Class Template Reference

```
#include <ql/Patterns/visitor.hpp>
```

### 7.808.1 Detailed Description

```
template<class T> class QuantLib::Visitor< T >
```

Visitor for a specific class

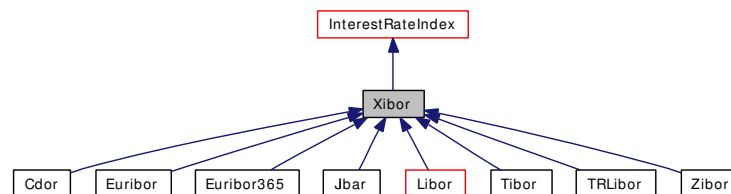
#### Public Member Functions

- virtual void **visit** (T &)=0

## 7.809 Xibor Class Reference

```
#include <ql/Indexes/xibor.hpp>
```

Inheritance diagram for Xibor:



### 7.809.1 Detailed Description

base class for LIBOR-like indexes

#### Todo

add methods returning [InterestRate](#)

### Public Member Functions

- **Xibor** (const std::string &familyName, const [Period](#) &tenor, [Integer](#) settlementDays, const [Currency](#) &currency, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

#### InterestRateIndex interface

- [Rate](#) **forecastFixing** (const [Date](#) &fixingDate) const
- boost::shared\_ptr< [YieldTermStructure](#) > **termStructure** () const

#### Inspectors

- [Frequency](#) **frequency** () const
- bool **isAdjusted** () const
- [BusinessDayConvention](#) **businessDayConvention** () const

#### Date calculations

- [Date](#) **maturityDate** (const [Date](#) &valueDate) const

### Protected Attributes

- [BusinessDayConvention](#) **convention\_**
- [Handle](#)< [YieldTermStructure](#) > **termStructure\_**

## 7.809.2 Member Function Documentation

### 7.809.2.1 [Frequency](#) frequency () const

#### [Deprecated](#)

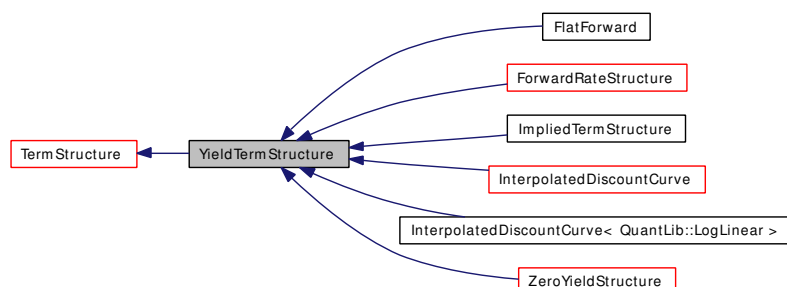
use tenor() instead



## 7.810 YieldTermStructure Class Reference

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

Inheritance diagram for YieldTermStructure:



### 7.810.1 Detailed Description

Interest-rate 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 continuous compounding.

#### Todo

add derived class ParSwapTermStructure similar to ZeroYieldTermStructure, Discount-Structure, [ForwardRateStructure](#)

#### Tests

observability against evaluation date changes is checked.

## Public Member Functions

### zero-yield rates

*These methods return the implied zero-yield rate for a given date or time. In the former case, the time is calculated as a fraction of year from the reference date.*

- [InterestRate](#) [zeroRate](#) (const [Date](#) &d, const [DayCounter](#) &resultDayCounter, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const
- [InterestRate](#) [zeroRate](#) (Time t, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const

### discount factors

*These methods return the discount factor for a given date or time. In the former case, the time is calculated as a fraction of year from the reference date.*

- [DiscountFactor](#) [discount](#) (const [Date](#) &, bool extrapolate=false) const
- [DiscountFactor](#) [discount](#) (Time, bool extrapolate=false) const

**forward rates**

*These methods returns the implied forward interest rate between two dates or times. In the former case, times are calculated as fractions of year from the reference date.*

- [InterestRate](#) [forwardRate](#) (const [Date](#) &d1, const [Date](#) &d2, const [DayCounter](#) &result-DayCounter, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const
- [InterestRate](#) [forwardRate](#) (Time t1, Time t2, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const

**par rates**

*These methods returns the implied par rate for a given sequence of payments at the given dates or times. In the former case, times are calculated as fractions of year from the reference date.*

**Warning**

*though somewhat related to a swap rate, this method is not to be used for the fair rate of a real swap, since it does not take into account all the market conventions' details. The correct way to evaluate such rate is to instantiate a [SimpleSwap](#) with the correct conventions, pass it the term structure and call the swap's [fairRate\(\)](#) method.*

- [Rate](#) [parRate](#) (Integer tenor, const [Date](#) &startDate, [Frequency](#) freq=Annual, bool extrapolate=false) const
- [Rate](#) [parRate](#) (const std::vector< [Date](#) > &dates, [Frequency](#) freq=Annual, bool extrapolate=false) const
- [Rate](#) [parRate](#) (const std::vector< Time > &times, [Frequency](#) freq=Annual, bool extrapolate=false) const

**Protected Member Functions****Calculations**

*These methods must be implemented in derived classes to perform the actual discount and rate calculations. When they are called, range check has already been performed; therefore, they must assume that extrapolation is required.*

- virtual [DiscountFactor](#) [discountImpl](#) (Time) const=0  
*discount calculation*

**7.810.2 Constructor & Destructor Documentation****7.810.2.1 [YieldTermStructure](#) ()**

default constructor

**Warning**

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

7.810.2.2 [YieldTermStructure](#) ()

default constructor

**Warning**

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

## 7.810.3 Member Function Documentation

7.810.3.1 [InterestRate](#) zeroRate (const [Date](#) & *d*, const [DayCounter](#) & *resultDayCounter*, Compounding *comp*, [Frequency](#) *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the required daycounting rule.

7.810.3.2 [InterestRate](#) zeroRate (Time *t*, Compounding *comp*, [Frequency](#) *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the same day-counting rule used by the term structure. The same rule should be used for calculating the passed time *t*.

7.810.3.3 [DiscountFactor](#) discount (Time, bool *extrapolate* = false) const

The same day-counting rule used by the term structure should be used for calculating the passed time *t*.

7.810.3.4 [InterestRate](#) forwardRate (const [Date](#) & *d1*, const [Date](#) & *d2*, const [DayCounter](#) & *resultDayCounter*, Compounding *comp*, [Frequency](#) *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the required day-counting rule.

7.810.3.5 [InterestRate](#) forwardRate (Time *t1*, Time *t2*, Compounding *comp*, [Frequency](#) *freq* = Annual, bool *extrapolate* = false) const

The resulting interest rate has the same day-counting rule used by the term structure. The same rule should be used for the calculating the passed times *t1* and *t2*.

7.810.3.6 [Rate](#) parRate (const std::vector< [Date](#) > & *dates*, [Frequency](#) *freq* = Annual, bool *extrapolate* = false) const

the first date in the vector must equal the start date; the following dates must equal the payment dates.

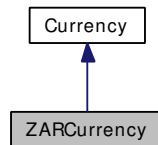
7.810.3.7 **Rate** `parRate (const std::vector< Time > & times, Frequency freq = Annual, bool extrapolate = false) const`

the first time in the vector must equal the start time; the following times must equal the payment times.

## 7.811 ZARCurrency Class Reference

```
#include <ql/Currencies/africa.hpp>
```

Inheritance diagram for ZARCurrency:



### 7.811.1 Detailed Description

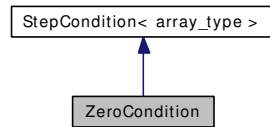
South-African rand.

The ISO three-letter code is ZAR; the numeric code is 710. It is divided into 100 cents.

## 7.812 ZeroCondition Class Template Reference

```
#include <ql/FiniteDifferences/zerocondition.hpp>
```

Inheritance diagram for ZeroCondition:



### 7.812.1 Detailed Description

```
template<class array_type> class QuantLib::ZeroCondition< array_type >
```

Zero exercise condition.

Used in CEV models

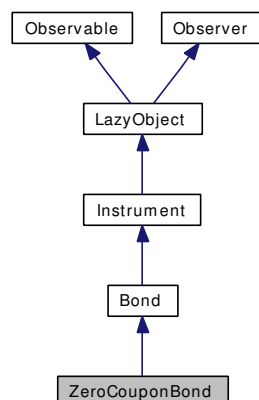
#### Public Member Functions

- void **applyTo** (array\_type &a, Time) const

## 7.813 ZeroCouponBond Class Reference

```
#include <ql/Instruments/zerocouponbond.hpp>
```

Inheritance diagram for ZeroCouponBond:



### 7.813.1 Detailed Description

zero-coupon bond

#### Tests

calculations are tested by checking results against cached values.

### Public Member Functions

- **ZeroCouponBond** (*Real* faceAmount, const *Date* &issueDate, const *Date* &maturityDate, *Integer* settlementDays, const *DayCounter* &dayCounter, const *Calendar* &calendar, *BusinessDayConvention* paymentConvention=Following, *Real* redemption=100.0, const *Handle*< *YieldTermStructure* > &discountCurve=*Handle*< *YieldTermStructure* >())
- **ZeroCouponBond** (const *Date* &issueDate, const *Date* &maturityDate, *Integer* settlementDays, const *DayCounter* &dayCounter, const *Calendar* &calendar, *BusinessDayConvention* paymentConvention=Following, *Real* redemption=100.0, const *Handle*< *YieldTermStructure* > &discountCurve=*Handle*< *YieldTermStructure* >())

### 7.813.2 Constructor & Destructor Documentation

- 7.813.2.1 **ZeroCouponBond** (const *Date* & issueDate, const *Date* & maturityDate, *Integer* settlementDays, const *DayCounter* & dayCounter, const *Calendar* & calendar, *BusinessDayConvention* paymentConvention = Following, *Real* redemption = 100.0, const *Handle*< *YieldTermStructure* > & discountCurve = *Handle*< *YieldTermStructure* >())

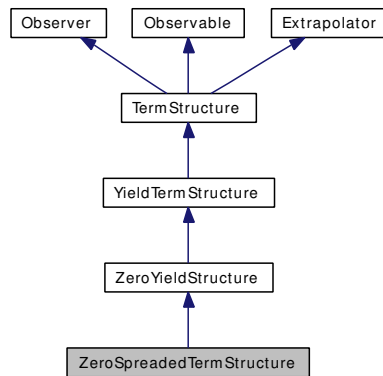
#### Deprecated

use constructor with face amount instead

## 7.814 ZeroSpreadedTermStructure Class Reference

```
#include <ql/TermStructures/zerospreadedtermstructure.hpp>
```

Inheritance diagram for ZeroSpreadedTermStructure:



### 7.814.1 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.

#### Tests

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

### Public Member Functions

- **ZeroSpreadedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Handle](#)< [Quote](#) > &spread)

#### YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Calendar](#) [calendar](#) () const  
*the calendar used for reference date calculation*
- const [Date](#) & [referenceDate](#) () const  
*the date at which discount = 1.0 and/or variance = 0.0*



- [Date maxDate](#) () const  
*the latest date for which the curve can return values*
- [Time maxTime](#) () const  
*the latest time for which the curve can return values*

### Protected Member Functions

- [Rate zeroYieldImpl](#) (Time) const  
*returns the spreaded zero yield rate*
- [Rate forwardImpl](#) (Time) const  
*returns the spreaded forward rate*

## 7.815 ZeroYield Struct Reference

```
#include <ql/TermStructures/bootstraptraits.hpp>
```

### 7.815.1 Detailed Description

Zero-curve traits.

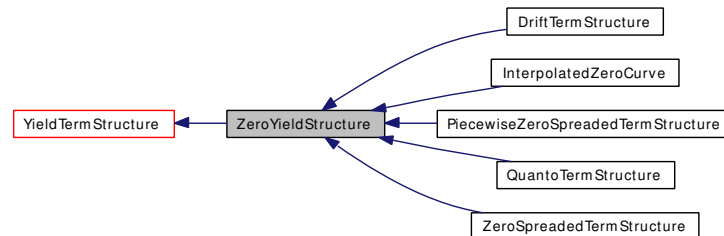
#### Static Public Member Functions

- static [Rate](#) **initialValue** ()
- static [Rate](#) **initialGuess** ()
- static [Rate](#) **guess** (const [YieldTermStructure](#) \*c, const [Date](#) &d)
- static [Rate](#) **minValueAfter** (Size, const std::vector< [Real](#) > &)
- static [Rate](#) **maxValueAfter** (Size, const std::vector< [Real](#) > &)
- static void **updateGuess** (std::vector< [Rate](#) > &data, [Rate](#) rate, Size i)

## 7.816 ZeroYieldStructure Class Reference

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

Inheritance diagram for ZeroYieldStructure:



### 7.816.1 Detailed Description

Zero-yield term structure.

This abstract class acts as an adapter to [YieldTermStructure](#) allowing the programmer to implement only the `zeroYieldImpl(Time, bool)` method in derived classes. [Discount](#) and forward are calculated from zero yields.

Rates are assumed to be annual continuous compounding.

### Protected Member Functions

#### YieldTermStructure implementation

- [DiscountFactor](#) `discountImpl` (Time) const
- virtual [Rate](#) `zeroYieldImpl` (Time) const=0  
*zero-yield calculation*

### 7.816.2 Member Function Documentation

#### 7.816.2.1 [DiscountFactor](#) `discountImpl` (Time) const [protected, virtual]

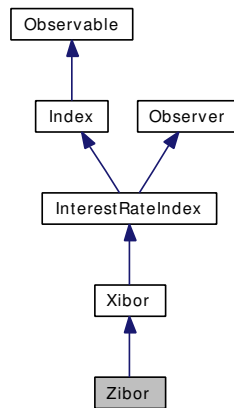
Returns the discount factor for the given date calculating it from the zero yield.

Implements [YieldTermStructure](#).

## 7.817 Zibor Class Reference

```
#include <ql/Indexes/zibor.hpp>
```

Inheritance diagram for Zibor:



### 7.817.1 Detailed Description

CHF ZIBOR rate

Zurich Interbank Offered Rate.

#### Warning

This is the rate fixed in Zurich by BBA. Use [CHFLibor](#) if you're interested in the London fixing by BBA.

#### Todo

check settlement days and day-count.

### Public Member Functions

- **Zibor** (const [Period](#) &tenor, const [Handle](#)< [YieldTermStructure](#) > &h=[Handle](#)< [YieldTermStructure](#) >())

## Chapter 8

# QuantLib File Documentation

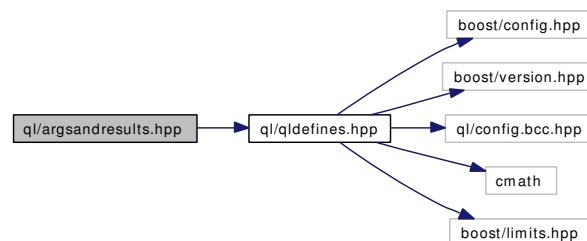
### 8.1 ql/argsandresults.hpp File Reference

#### 8.1.1 Detailed Description

Base classes for generic arguments and results.

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

Include dependency graph for argsandresults.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Arguments](#)  
*base class for generic argument groups*
- class [Results](#)  
*base class for generic result groups*

## Defines

- `#define QL_MIN_VOLATILITY 1.0e-7`
- `#define QL_MIN_DIVYIELD 1.0e-7`
- `#define QL_MAX_VOLATILITY 4.0`
- `#define QL_MAX_DIVYIELD 4.0`

## 8.2 ql/calendar.hpp File Reference

### 8.2.1 Detailed Description

calendar class

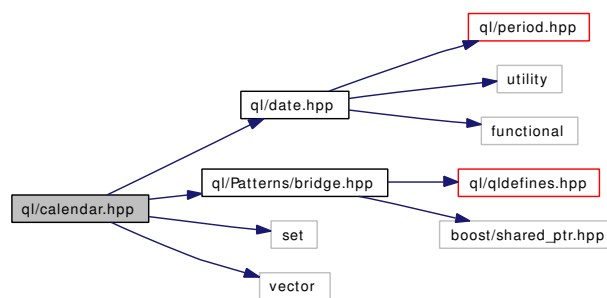
```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/bridge.hpp>
```

```
#include <set>
```

```
#include <vector>
```

Include dependency graph for calendar.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **CalendarImpl**  
*abstract base class for calendar implementations*
- class **Calendar**  
*calendar class*
- class **Calendar::WesternImpl**  
*partial calendar implementation*
- class **Calendar::OrthodoxImpl**  
*partial calendar implementation*

## Enumerations

- enum **BusinessDayConvention** {  
Following, ModifiedFollowing, Preceding, ModifiedPreceding,  
Unadjusted, MonthEndReference, UnadjustedMonthEnd }

*Business Day conventions.*



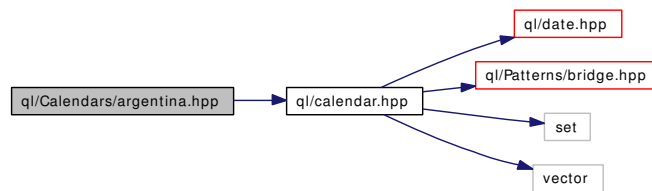
## 8.3 ql/Calendars/argentina.hpp File Reference

### 8.3.1 Detailed Description

Argentinian calendars.

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

Include dependency graph for argentina.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Argentina](#)  
*Argentinian calendars.*

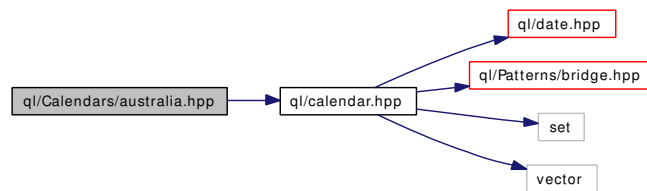
## 8.4 ql/Calendars/australia.hpp File Reference

### 8.4.1 Detailed Description

Australian calendar.

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

Include dependency graph for australia.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Australia](#)  
*Australian calendar.*

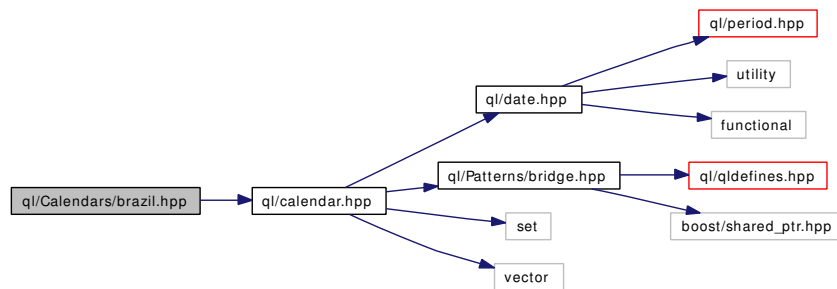
## 8.5 ql/Calendars/brazil.hpp File Reference

### 8.5.1 Detailed Description

Brazilian calendar.

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

Include dependency graph for brazil.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Brazil](#)  
*Brazilian calendar.*

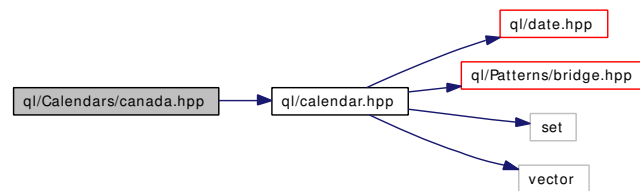
## 8.6 ql/Calendars/canada.hpp File Reference

### 8.6.1 Detailed Description

Canadian calendar.

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

Include dependency graph for canada.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Canada](#)  
*Canadian calendar.*

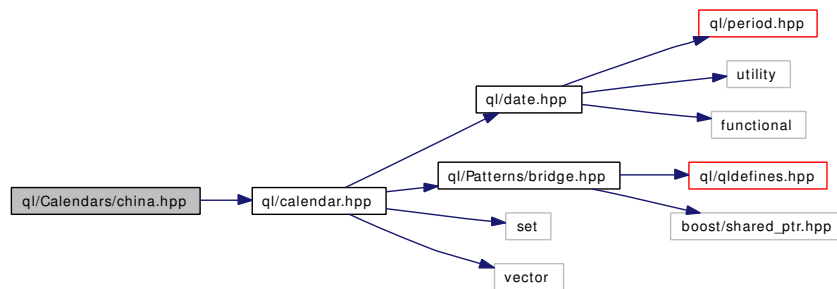
## 8.7 ql/Calendars/china.hpp File Reference

### 8.7.1 Detailed Description

Chinese calendar.

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

Include dependency graph for china.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [China](#)  
*Chinese calendar.*

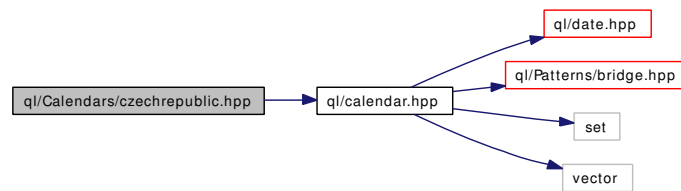
## 8.8 ql/Calendars/czechrepublic.hpp File Reference

### 8.8.1 Detailed Description

Czech calendars.

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

Include dependency graph for czechrepublic.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CzechRepublic](#)  
*Czech calendars.*

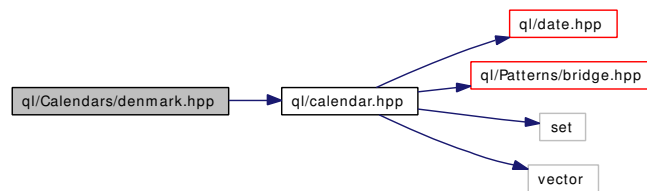
## 8.9 ql/Calendars/denmark.hpp File Reference

### 8.9.1 Detailed Description

Danish calendar.

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

Include dependency graph for `denmark.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Denmark](#)  
*Danish calendar.*

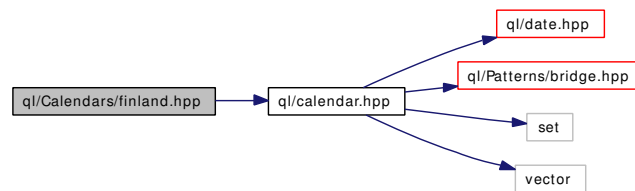
## 8.10 ql/Calendars/finland.hpp File Reference

### 8.10.1 Detailed Description

Finnish calendar.

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

Include dependency graph for finland.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Finland](#)  
*Finnish calendar.*



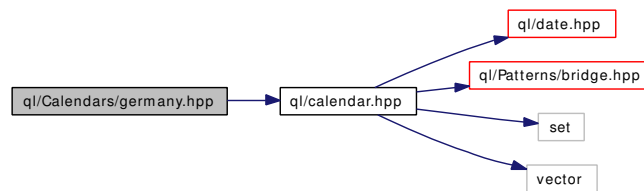
## 8.11 ql/Calendars/germany.hpp File Reference

### 8.11.1 Detailed Description

German calendars.

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

Include dependency graph for germany.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Germany](#)  
*German calendars.*

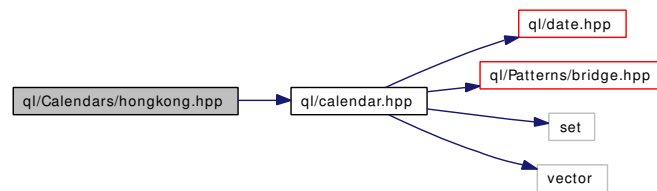
## 8.12 ql/Calendars/hongkong.hpp File Reference

### 8.12.1 Detailed Description

Hong Kong calendars.

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

Include dependency graph for hongkong.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [HongKong](#)  
*Hong Kong calendars.*

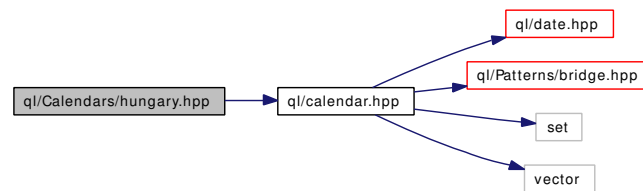
## 8.13 ql/Calendars/hungary.hpp File Reference

### 8.13.1 Detailed Description

Hungarian calendar.

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

Include dependency graph for hungary.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Hungary**  
*Hungarian calendar.*

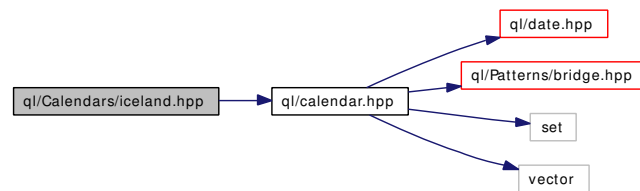
## 8.14 ql/Calendars/iceland.hpp File Reference

### 8.14.1 Detailed Description

Icelandic calendars.

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

Include dependency graph for iceland.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Iceland](#)  
*Icelandic calendars.*

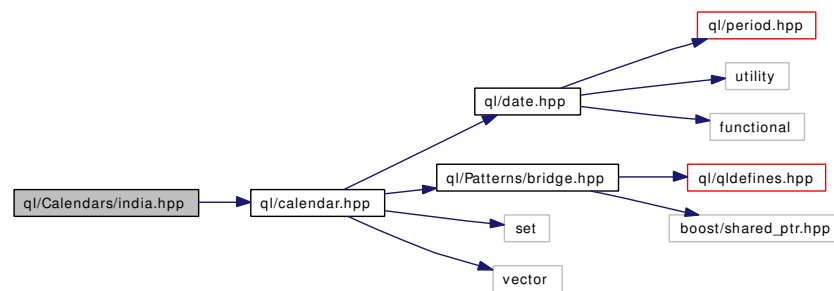
## 8.15 ql/Calendars/india.hpp File Reference

### 8.15.1 Detailed Description

Indian calendars.

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

Include dependency graph for india.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [India](#)  
*Indian calendars.*

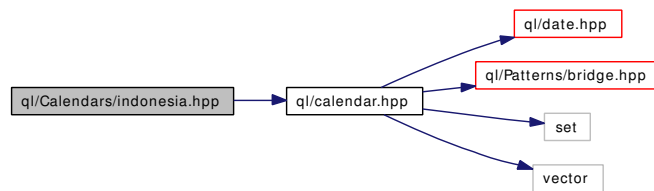
## 8.16 ql/Calendars/indonesia.hpp File Reference

### 8.16.1 Detailed Description

Indonesian calendars.

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

Include dependency graph for indonesia.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Indonesia](#)  
*Indonesian calendars*

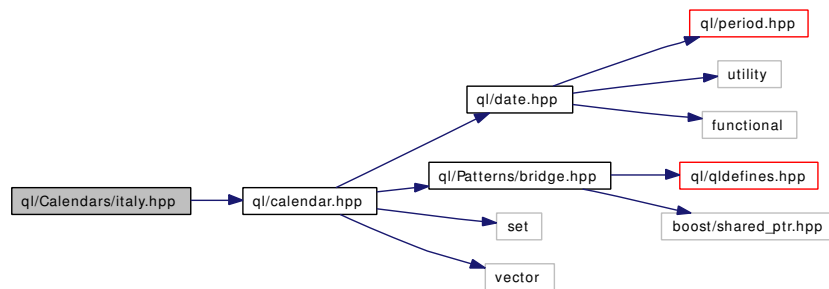
## 8.17 ql/Calendars/italy.hpp File Reference

### 8.17.1 Detailed Description

Italian calendars.

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

Include dependency graph for italy.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Italy](#)  
*Italian calendars.*

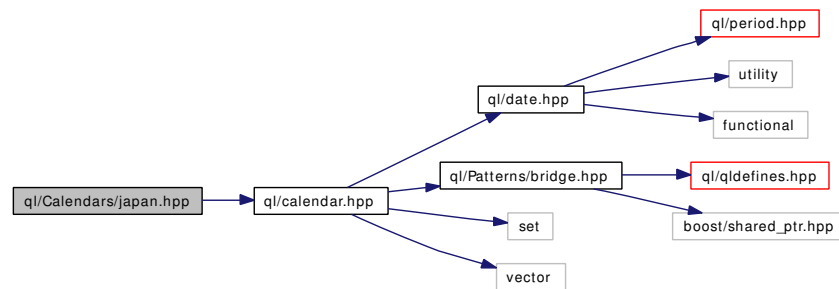
## 8.18 ql/Calendars/japan.hpp File Reference

### 8.18.1 Detailed Description

Japanese calendar.

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

Include dependency graph for japan.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Japan](#)  
*Japanese calendar.*



## 8.19 ql/Calendars/jointcalendar.hpp File Reference

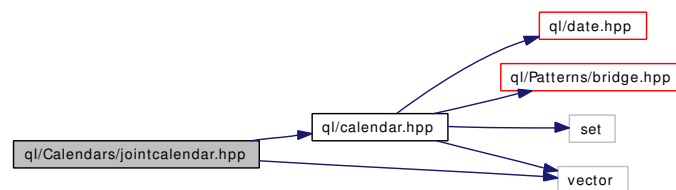
### 8.19.1 Detailed Description

Joint calendar.

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

```
#include <vector>
```

Include dependency graph for jointcalendar.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [JointCalendar](#)  
*Joint calendar.*

### Enumerations

- enum **JointCalendarRule** { [JoinHolidays](#), [JoinBusinessDays](#) }  
*rules for joining calendars*

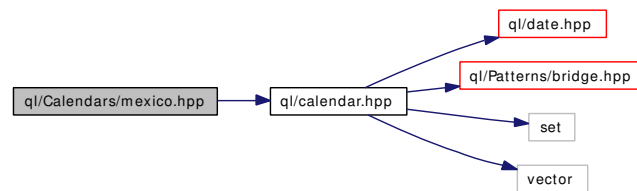
## 8.20 ql/Calendars/mexico.hpp File Reference

### 8.20.1 Detailed Description

Mexican calendars.

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

Include dependency graph for mexico.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Mexico](#)  
*Mexican calendars*

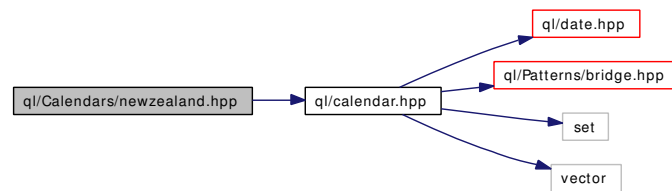
## 8.21 ql/Calendars/newzealand.hpp File Reference

### 8.21.1 Detailed Description

New Zealand calendar.

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

Include dependency graph for newzealand.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [NewZealand](#)  
*New Zealand calendar.*

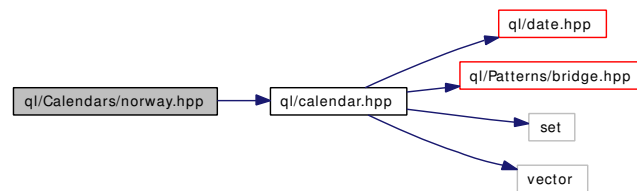
## 8.22 ql/Calendars/norway.hpp File Reference

### 8.22.1 Detailed Description

Norwegian calendar.

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

Include dependency graph for norway.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Norway](#)  
*Norwegian calendar.*

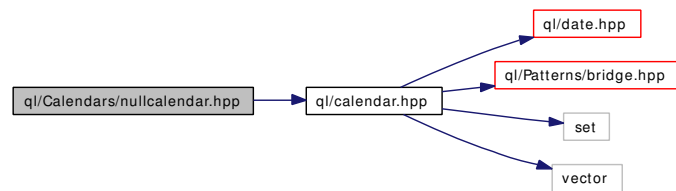
## 8.23 ql/Calendars/nullcalendar.hpp File Reference

### 8.23.1 Detailed Description

Calendar for reproducing theoretical calculations.

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

Include dependency graph for nullcalendar.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [NullCalendar](#)  
*Calendar for reproducing theoretical calculations.*

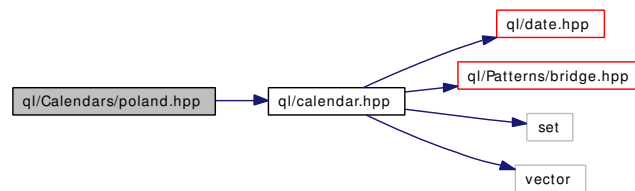
## 8.24 ql/Calendars/poland.hpp File Reference

### 8.24.1 Detailed Description

Polish calendar.

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

Include dependency graph for poland.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Poland](#)  
*Polish calendar.*

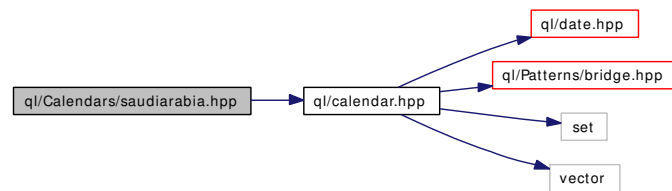
## 8.25 ql/Calendars/saudiarabia.hpp File Reference

### 8.25.1 Detailed Description

Saudi Arabian calendar.

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

Include dependency graph for saudiarabia.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SaudiArabia](#)  
*Saudi Arabian calendar.*

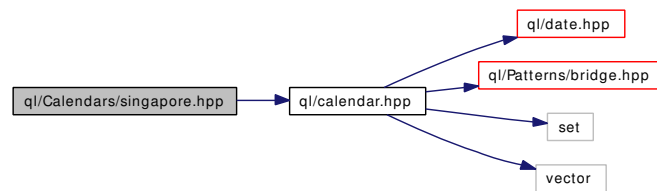
## 8.26 ql/Calendars/singapore.hpp File Reference

### 8.26.1 Detailed Description

Singapore calendars.

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

Include dependency graph for singapore.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Singapore](#)  
*Singapore calendars*



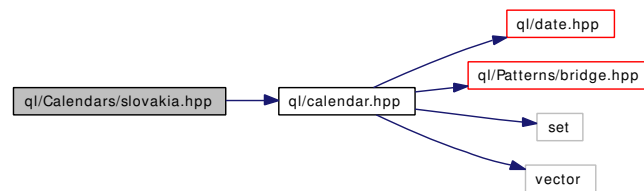
## 8.27 ql/Calendars/slovakia.hpp File Reference

### 8.27.1 Detailed Description

Slovak calendars.

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

Include dependency graph for `slovakia.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Slovakia](#)  
*Slovak calendars.*

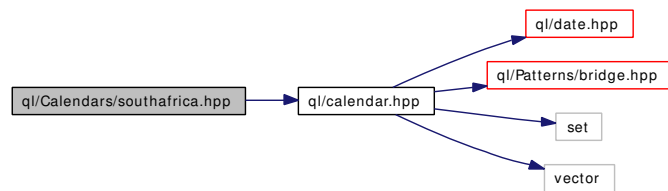
## 8.28 ql/Calendars/southafrica.hpp File Reference

### 8.28.1 Detailed Description

South-African calendar.

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

Include dependency graph for southafrica.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SouthAfrica](#)  
*South-African calendar.*

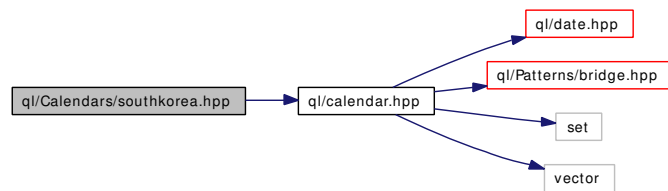
## 8.29 ql/Calendars/southkorea.hpp File Reference

### 8.29.1 Detailed Description

South Korean calendars.

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

Include dependency graph for southkorea.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SouthKorea](#)  
*South Korean calendars.*

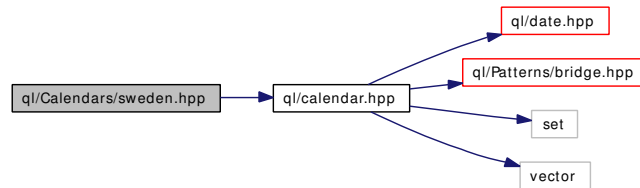
## 8.30 ql/Calendars/sweden.hpp File Reference

### 8.30.1 Detailed Description

Swedish calendar.

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

Include dependency graph for sweden.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Sweden](#)  
*Swedish calendar.*

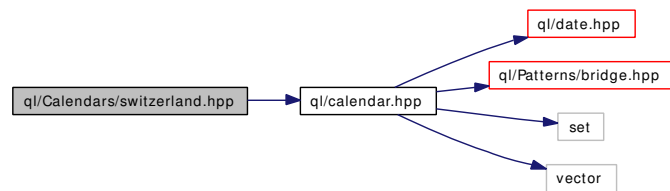
## 8.31 ql/Calendars/switzerland.hpp File Reference

### 8.31.1 Detailed Description

Swiss calendar.

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

Include dependency graph for switzerland.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Switzerland](#)  
*Swiss calendar.*

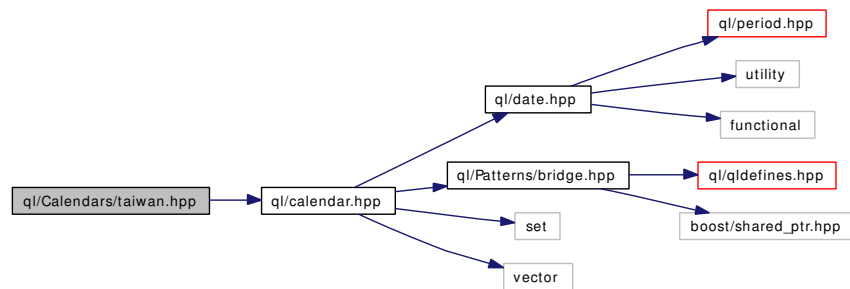
## 8.32 ql/Calendars/taiwan.hpp File Reference

### 8.32.1 Detailed Description

Taiwanese calendars.

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

Include dependency graph for taiwan.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Taiwan](#)  
*Taiwanese calendars.*

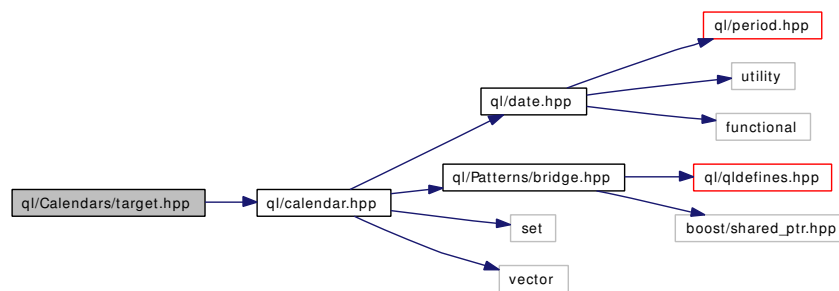
## 8.33 ql/Calendars/target.hpp File Reference

### 8.33.1 Detailed Description

TARGET calendar.

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

Include dependency graph for target.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **TARGET**  
*TARGET calendar*

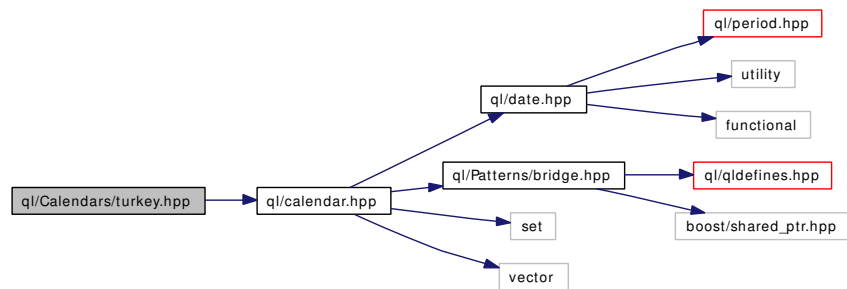
## 8.34 ql/Calendars/turkey.hpp File Reference

### 8.34.1 Detailed Description

Turkish calendar.

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

Include dependency graph for turkey.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Turkey](#)  
*Turkish calendar.*



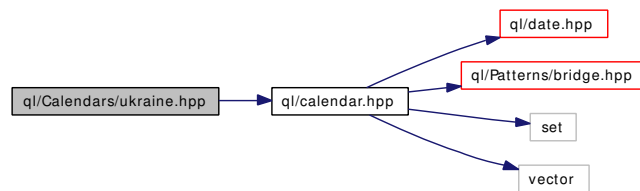
## 8.35 ql/Calendars/ukraine.hpp File Reference

### 8.35.1 Detailed Description

Ukrainian calendars.

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

Include dependency graph for ukraine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Ukraine](#)  
*Ukrainian calendars.*

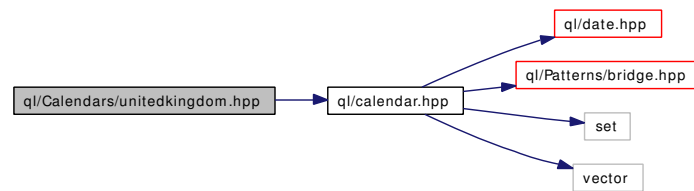
## 8.36 ql/Calendars/unitedkingdom.hpp File Reference

### 8.36.1 Detailed Description

UK calendars.

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

Include dependency graph for unitedkingdom.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **UnitedKingdom**  
*United Kingdom calendars.*

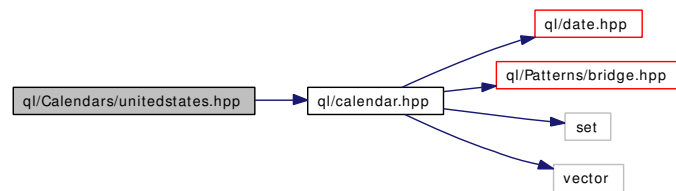
## 8.37 ql/Calendars/unitedstates.hpp File Reference

### 8.37.1 Detailed Description

US calendars.

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

Include dependency graph for unitedstates.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **UnitedStates**  
*United States calendars.*

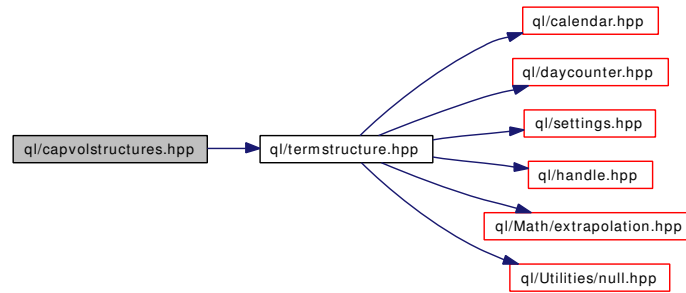
## 8.38 ql/capvolstructures.hpp File Reference

### 8.38.1 Detailed Description

Cap/Floor volatility structures.

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

Include dependency graph for capvolstructures.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **CapVolatilityStructure**  
*Cap/floor term-volatility structure.*
- class **CapletVolatilityStructure**  
*Caplet/floorlet forward-volatility structure.*

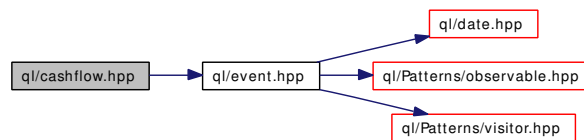
## 8.39 ql/cashflow.hpp File Reference

### 8.39.1 Detailed Description

Base class for cash flows.

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

Include dependency graph for cashflow.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CashFlow](#)  
*Base class for cash flows.*

## 8.40 ql/CashFlows/analysis.hpp File Reference

### 8.40.1 Detailed Description

Cash-flow analysis functions.

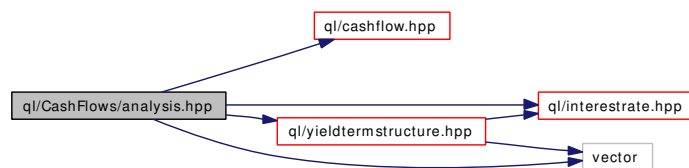
```
#include <ql/cashflow.hpp>
```

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

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

```
#include <vector>
```

Include dependency graph for analysis.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct [Duration](#)  
*duration type*
- class [Cashflows](#)  
*cashflows analysis functions*

## 8.41 ql/CashFlows/cashflowvectors.hpp File Reference

### 8.41.1 Detailed Description

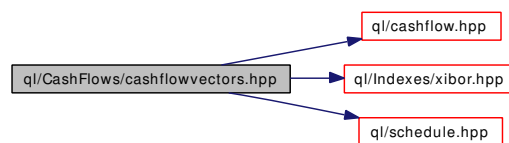
Cash flow vector builders.

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

```
#include <ql/Indexes/xibor.hpp>
```

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

Include dependency graph for cashflowvectors.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- `std::vector< boost::shared_ptr< CashFlow > > FixedRateCouponVector` (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const std::vector< Rate > &couponRates, const DayCounter &dayCount, const DayCounter &firstPeriodDayCount=DayCounter())  
*helper function building a sequence of fixed rate coupons*
- `std::vector< boost::shared_ptr< CashFlow > > FloatingRateCouponVector` (const Schedule &schedule, const BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const Integer fixingDays, const boost::shared\_ptr< Xibor > &index, const std::vector< Real > &gearings=std::vector< Real >(), const std::vector< Spread > &spreads=std::vector< Spread >(), const DayCounter &dayCounter=DayCounter())  
*helper function building a sequence of par coupons*

## 8.42 ql/CashFlows/cmscoupon.hpp File Reference

### 8.42.1 Detailed Description

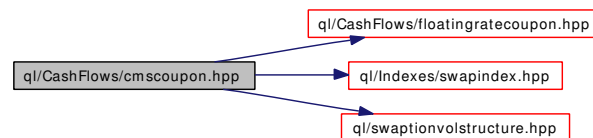
CMS coupon.

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

```
#include <ql/Indexes/swapindex.hpp>
```

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

Include dependency graph for cmscoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [VanillaCMSCouponPricer](#)  
*pricer for vanilla CMS coupons*
- class [CMSCoupon](#)  
*CMS coupon class.*

### Functions

- `std::vector< boost::shared_ptr< CashFlow > > CMSCouponVector (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const boost::shared_ptr< SwapIndex > &index, Integer fixingDays, const DayCounter &dayCounter, const std::vector< Real > &baseRate, const std::vector< Real > &fractions, const std::vector< Real > &caps, const std::vector< Real > &floors, const std::vector< Real > &meanReversions, const boost::shared_ptr< VanillaCMSCouponPricer > &pricer, const Handle< SwaptionVolatilityStructure > &vol=Handle< SwaptionVolatilityStructure >())`
- `std::vector< boost::shared_ptr< CashFlow > > CMSZeroCouponVector (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const boost::shared_ptr< SwapIndex > &index, Integer fixingDays, const DayCounter &dayCounter, const std::vector< Real > &baseRate, const std::vector< Real > &fractions, const std::vector< Real > &caps, const std::vector< Real > &floors, const std::vector< Real > &meanReversions, const boost::shared_ptr< VanillaCMSCouponPricer > &pricer, const Handle< SwaptionVolatilityStructure > &vol=Handle< SwaptionVolatilityStructure >())`
- `std::vector< boost::shared_ptr< CashFlow > > CMSInArrearsCouponVector (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real >`



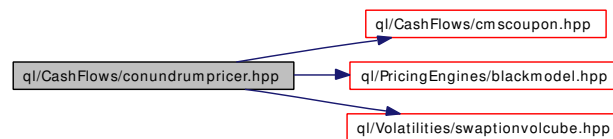
```
&nominals, const boost::shared_ptr< SwapIndex > &index, Integer fixingDays, const Day-  
Counter &dayCounter, const std::vector< Real > &baseRate, const std::vector< Real > &frac-  
tions, const std::vector< Real > &caps, const std::vector< Real > &floors, const std::vector<  
Real > &meanReversions, const boost::shared_ptr< VanillaCMSCouponPricer > &pricer,  
const Handle< SwaptionVolatilityStructure > &vol=Handle< SwaptionVolatilityStructure  
>())
```

## 8.43 ql/CashFlows/conundrumpricer.hpp File Reference

### 8.43.1 Detailed Description

```
#include <ql/CashFlows/cmscoupon.hpp>
#include <ql/PricingEngines/blackmodel.hpp>
#include <ql/Volatilities/swaptionvolcube.hpp>
```

Include dependency graph for conundrumpricer.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **ConundrumPricer**  
*ConundrumPricer.*
- class **ConundrumPricerByNumericalIntegration**  
*ConundrumPricerByNumericalIntegration.*

## 8.44 ql/CashFlows/coupon.hpp File Reference

### 8.44.1 Detailed Description

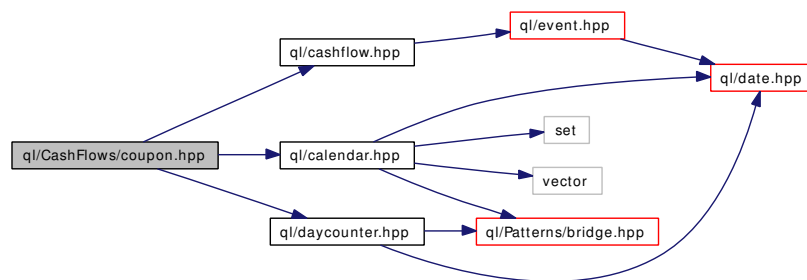
Coupon accruing over a fixed period.

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

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

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

Include dependency graph for coupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Coupon**  
*coupon accruing over a fixed period*

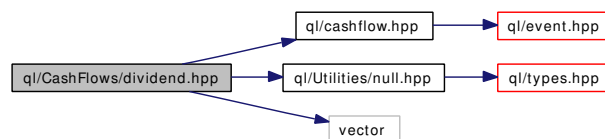
## 8.45 ql/CashFlows/dividend.hpp File Reference

### 8.45.1 Detailed Description

A stock dividend.

```
#include <ql/cashflow.hpp>
#include <ql/Utilities/null.hpp>
#include <vector>
```

Include dependency graph for dividend.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Dividend](#)  
*Predetermined cash flow.*
- class [FixedDividend](#)  
*Predetermined cash flow.*
- class [FractionalDividend](#)  
*Predetermined cash flow.*

### Functions

- `std::vector< boost::shared_ptr< Dividend > > DividendVector (const std::vector< Date > &dividendDates, const std::vector< Real > &dividends)`  
*helper function building a sequence of fixed dividends*

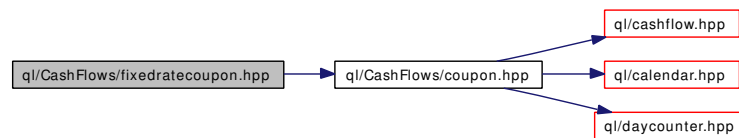
## 8.46 ql/CashFlows/fixedratecoupon.hpp File Reference

### 8.46.1 Detailed Description

Coupon paying a fixed annual rate.

```
#include <ql/CashFlows/coupon.hpp>
```

Include dependency graph for fixedratecoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **FixedRateCoupon**  
*Coupon paying a fixed interest rate*

## 8.47 ql/CashFlows/floatingratecoupon.hpp File Reference

### 8.47.1 Detailed Description

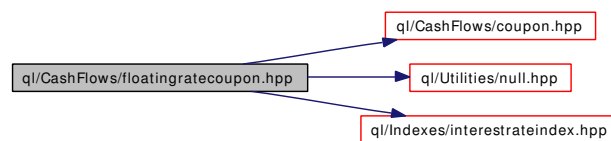
Coupon paying a variable index-based rate.

```
#include <ql/CashFlows/coupon.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

```
#include <ql/Indexes/interestrateindex.hpp>
```

Include dependency graph for floatingratecoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FloatingRateCoupon](#)  
*Coupon paying a variable index-based rate*

## 8.48 ql/CashFlows/inarrearindexedcoupon.hpp File Reference

### 8.48.1 Detailed Description

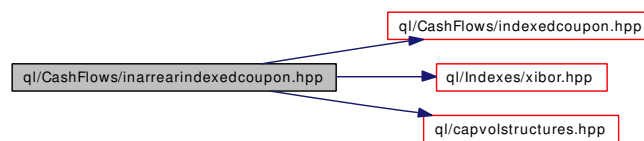
in-arrear floating-rate coupon

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

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

Include dependency graph for inarrearindexedcoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InArrearIndexedCoupon](#)  
*In-arrear floating-rate coupon.*

## 8.49 ql/CashFlows/indexedcashflowvectors.hpp File Reference

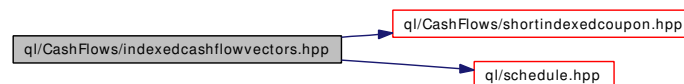
### 8.49.1 Detailed Description

Indexed cash-flow vector builders.

```
#include <ql/CashFlows/shortindexedcoupon.hpp>
```

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

Include dependency graph for indexedcashflowvectors.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- `template<class IndexedCouponType, class IndexType> std::vector< boost::shared_ptr< CashFlow > > IndexedCouponVector (const Schedule &schedule, const BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const Integer fixingDays, const boost::shared_ptr< IndexType > &index, const std::vector< Real > &gearings, const std::vector< Spread > &spreads, const DayCounter &dayCounter=DayCounter())`  
*helper function building a leg of floating coupons*



## 8.50 ql/CashFlows/indexedcoupon.hpp File Reference

### 8.50.1 Detailed Description

indexed coupon

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

Include dependency graph for indexedcoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef FloatingRateCoupon [IndexedCoupon](#)

## 8.51 ql/CashFlows/parcoupon.hpp File Reference

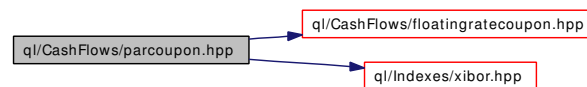
### 8.51.1 Detailed Description

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**

### Classes

- class **ParCoupon**  
*coupon at par on a term structure*

## 8.52 ql/CashFlows/shortfloatingcoupon.hpp File Reference

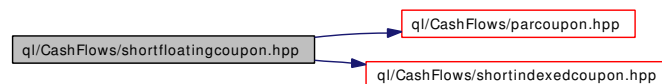
### 8.52.1 Detailed Description

Short (or long) coupon at par on a term structure.

```
#include <ql/CashFlows/parcoupon.hpp>
```

```
#include <ql/CashFlows/shortindexedcoupon.hpp>
```

Include dependency graph for shortfloatingcoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Short< ParCoupon >](#)  
*Short coupon at par on a term structure*

## 8.53 ql/CashFlows/shortindexedcoupon.hpp File Reference

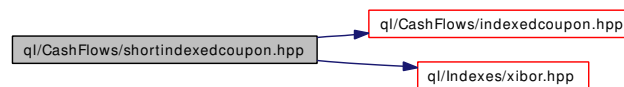
### 8.53.1 Detailed Description

Short (or long) indexed coupon.

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for shortindexedcoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Short](#)  
*Short indexed coupon*

## 8.54 ql/CashFlows/simplecashflow.hpp File Reference

### 8.54.1 Detailed Description

Predetermined cash flow.

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

Include dependency graph for simplecashflow.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SimpleCashFlow](#)  
*Predetermined cash flow.*

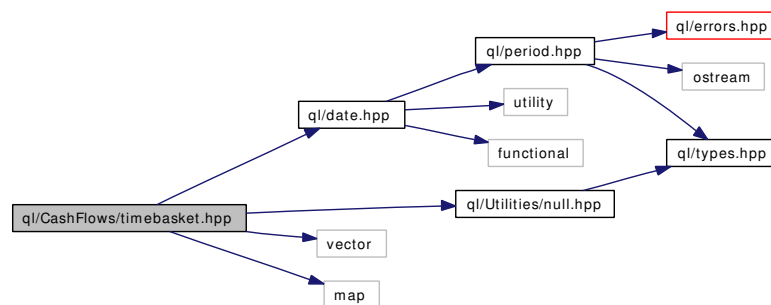
## 8.55 ql/CashFlows/timebasket.hpp File Reference

### 8.55.1 Detailed Description

Distribution over a number of dates

```
#include <ql/date.hpp>
#include <ql/Utilities/null.hpp>
#include <vector>
#include <map>
```

Include dependency graph for timebasket.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **TimeBasket**  
*Distribution over a number of dates.*

## 8.56 ql/CashFlows/upfrontindexedcoupon.hpp File Reference

### 8.56.1 Detailed Description

Up front indexed coupon.

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

Include dependency graph for upfrontindexedcoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **UpFrontIndexedCoupon**  
*up front indexed coupon class*

## 8.57 ql/Currencies/africa.hpp File Reference

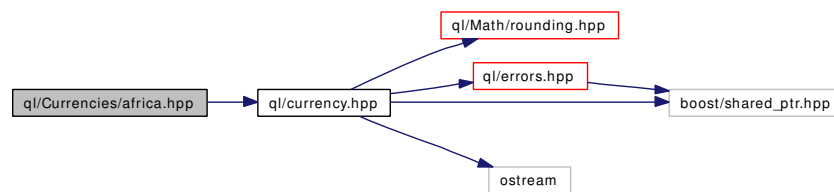
### 8.57.1 Detailed Description

African currencies.

Data from [http://fx.sauder.ubc.ca/currency\\_table.html](http://fx.sauder.ubc.ca/currency_table.html) and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

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

Include dependency graph for africa.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ZARCurrency](#)  
*South-African rand.*



## 8.58 ql/Currencies/america.hpp File Reference

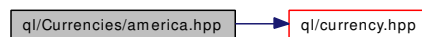
### 8.58.1 Detailed Description

American currencies.

Data from [http://fx.sauder.ubc.ca/currency\\_table.html](http://fx.sauder.ubc.ca/currency_table.html) and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

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

Include dependency graph for america.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ARSCurrency](#)  
*Argentinian peso.*
- class [BRLCurrency](#)  
*Brazilian real.*
- class [CADCurrency](#)  
*Canadian dollar.*
- class [CLPCurrency](#)  
*Chilean peso.*
- class [COPCurrency](#)  
*Colombian peso.*
- class [MXNCurrency](#)  
*Mexican peso.*
- class [TTDCurrency](#)  
*Trinidad & Tobago dollar.*
- class [USDCurrency](#)  
*U.S. dollar.*
- class [VEBCurrency](#)  
*Venezuelan bolivar.*

## 8.59 ql/Currencies/asia.hpp File Reference

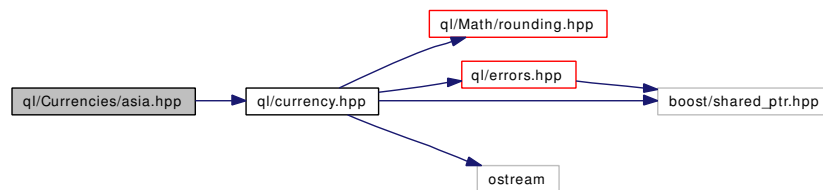
### 8.59.1 Detailed Description

Asian currencies.

Data from [http://fx.sauder.ubc.ca/currency\\_table.html](http://fx.sauder.ubc.ca/currency_table.html) and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

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

Include dependency graph for asia.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BDTCurrency**  
*Bangladesh taka.*
- class **CNYCurrency**  
*Chinese yuan.*
- class **HKDCurrency**  
*Honk Kong dollar.*
- class **ILSCurrency**  
*Israeli shekel.*
- class **INRCurrency**  
*Indian rupee.*
- class **IQDCurrency**  
*Iraqi dinar.*
- class **IRRCurrency**  
*Iranian rial.*
- class **JPYCurrency**  
*Japanese yen.*

- class [KRWCurrency](#)  
*South-Korean won.*
- class [KWDCurrency](#)  
*Kuwaiti dinar.*
- class [NPRCurrency](#)  
*Nepal rupee.*
- class [PKRCurrency](#)  
*Pakistani rupee.*
- class [SARCurrency](#)  
*Saudi riyal.*
- class [SGDCurrency](#)  
*Singapore dollar.*
- class [THBCurrency](#)  
*Thai baht.*
- class [TWDCurrency](#)  
*Taiwan dollar.*

## 8.60 ql/Currencies/europe.hpp File Reference

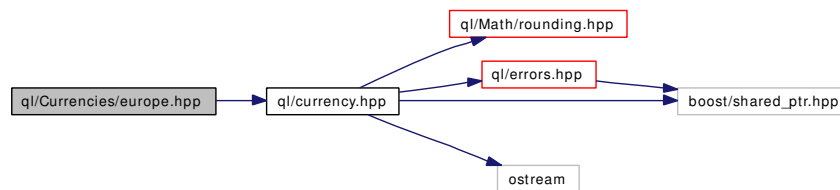
### 8.60.1 Detailed Description

European currencies.

Data from [http://fx.sauder.ubc.ca/currency\\_table.html](http://fx.sauder.ubc.ca/currency_table.html) and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

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

Include dependency graph for europe.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BGLCurrency**  
*Bulgarian lev.*
- class **BYRCurrency**  
*Belarussian ruble.*
- class **CHFCurrency**  
*Swiss franc.*
- class **CYPCurrency**  
*Cyprus pound.*
- class **CZKCurrency**  
*Czech koruna.*
- class **DKKCurrency**  
*Danish krone.*
- class **EEKCurrency**  
*Estonian kroon.*
- class **EURCurrency**  
*European Euro.*

- class [GBPCurrency](#)  
*British pound sterling.*
- class [HUFCurrency](#)  
*Hungarian forint.*
- class [ISKCurrency](#)  
*Iceland krona.*
- class [LTLCurrency](#)  
*Lithuanian litas.*
- class [LVLCurrency](#)  
*Latvian lat.*
- class [MTLCurrency](#)  
*Maltese lira.*
- class [NOKCurrency](#)  
*Norwegian krone.*
- class [PLNCurrency](#)  
*Polish zloty.*
- class [ROLCurrency](#)  
*Romanian leu.*
- class [RONCurrency](#)  
*Romanian new leu.*
- class [SEKCurrency](#)  
*Swedish krona.*
- class [SITCurrency](#)  
*Slovenian tolar.*
- class [SKKCurrency](#)  
*Slovak koruna.*
- class [TRLCurrency](#)  
*Turkish lira.*
- class [TRYCurrency](#)  
*New Turkish lira.*
- class [ATSCurrency](#)  
*Austrian shilling.*
- class [BEFCurrency](#)  
*Belgian franc.*

- class [DEMCurrency](#)  
*Deutsche mark.*
- class [ESPCurrency](#)  
*Spanish peseta.*
- class [FIMCurrency](#)  
*Finnish markka.*
- class [FRFCurrency](#)  
*French franc.*
- class [GRDCurrency](#)  
*Greek drachma.*
- class [IEPCurrency](#)  
*Irish punt.*
- class [ITLCurrency](#)  
*Italian lira.*
- class [LUFCurrency](#)  
*Luxembourg franc.*
- class [NLGCurrency](#)  
*Dutch guilder.*
- class [PTECurrency](#)  
*Portuguese escudo.*

## 8.61 ql/Currencies/exchangeratemanager.hpp File Reference

### 8.61.1 Detailed Description

exchange-rate repository

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

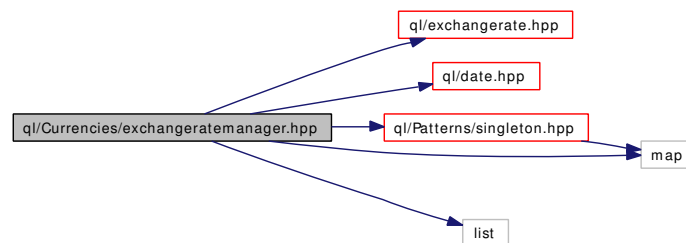
```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <list>
```

```
#include <map>
```

Include dependency graph for `exchangeratemanager.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ExchangeRateManager](#)  
*exchange-rate repository*

## 8.62 ql/Currencies/oceania.hpp File Reference

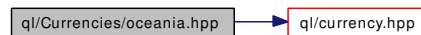
### 8.62.1 Detailed Description

Oceanian currencies.

Data from [http://fx.sauder.ubc.ca/currency\\_table.html](http://fx.sauder.ubc.ca/currency_table.html) and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

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

Include dependency graph for oceania.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **AUDCurrency**  
*Australian dollar.*
- class **NZDCurrency**  
*New Zealand dollar.*



## 8.63 ql/currency.hpp File Reference

### 8.63.1 Detailed Description

Known currencies.

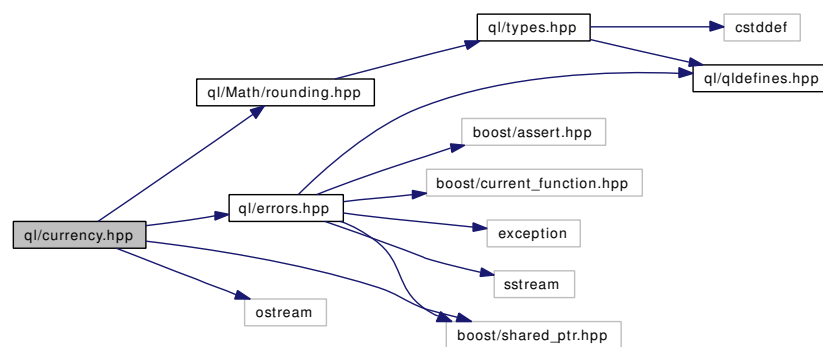
```
#include <ql/Math/rounding.hpp>
```

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

```
#include <boost/shared_ptr.hpp>
```

```
#include <ostream>
```

Include dependency graph for currency.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Currency**  
*Currency specification*

## 8.64 ql/date.hpp File Reference

### 8.64.1 Detailed Description

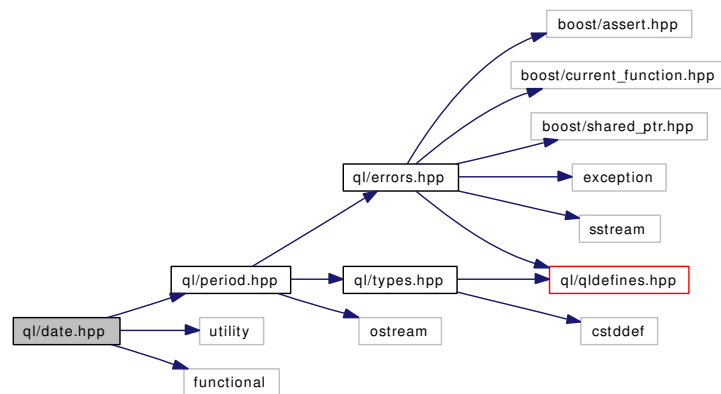
date- and time-related classes, typedefs and enumerations

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

```
#include <utility>
```

```
#include <functional>
```

Include dependency graph for date.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

### Classes

- struct **IMM**  
*Main cycle of the International **Money** Market (a.k.a. **IMM**) Months.*
- class **Date**  
*Concrete date class.*

### Typedefs

- typedef Integer **Day**  
*Day number.*
- typedef Integer **Year**  
*Year number.*

## Enumerations

- enum [Weekday](#) {  
**Sunday** = 1, **Monday** = 2, **Tuesday** = 3, **Wednesday** = 4,  
**Thursday** = 5, **Friday** = 6, **Saturday** = 7, **Sun** = 1,  
**Mon** = 2, **Tue** = 3, **Wed** = 4, **Thu** = 5,  
**Fri** = 6, **Sat** = 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,  
**Jan** = 1, **Feb** = 2, **Mar** = 3, **Apr** = 4,  
**Jun** = 6, **Jul** = 7, **Aug** = 8, **Sep** = 9,  
**Oct** = 10, **Nov** = 11, **Dec** = 12 }  
*Month names.*

## Functions

- std::ostream & **operator**<< (std::ostream &, const long\_weekday\_holder &)
- std::ostream & **operator**<< (std::ostream &, const short\_weekday\_holder &)
- std::ostream & **operator**<< (std::ostream &, const shortest\_weekday\_holder &)
- detail::long\_weekday\_holder [long\\_weekday](#) (Weekday)  
*output weekdays in long format*
- detail::short\_weekday\_holder [short\\_weekday](#) (Weekday)  
*output weekdays in short format (three letters)*
- detail::shortest\_weekday\_holder [shortest\\_weekday](#) (Weekday)  
*output weekdays in shortest format (two letters)*
- std::ostream & **operator**<< (std::ostream &, const short\_date\_holder &)
- std::ostream & **operator**<< (std::ostream &, const long\_date\_holder &)
- std::ostream & **operator**<< (std::ostream &, const iso\_date\_holder &)
- detail::short\_date\_holder [short\\_date](#) (const Date &)  
*output dates in short format (mm/dd/yyyy)*
- detail::long\_date\_holder [long\\_date](#) (const Date &)  
*output dates in long format (Month ddth, yyyy)*
- detail::iso\_date\_holder [iso\\_date](#) (const Date &)  
*output dates in ISO format (yyyy-mm-dd)*
- BigInteger **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)

## 8.65 ql/daycounter.hpp File Reference

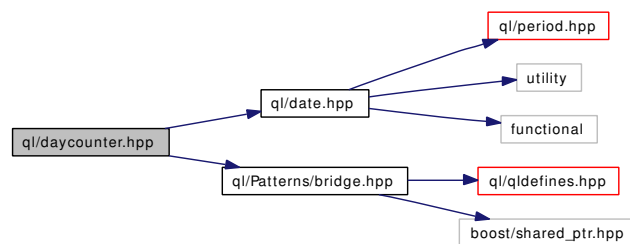
### 8.65.1 Detailed Description

day counter class

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

```
#include <ql/Patterns/bridge.hpp>
```

Include dependency graph for daycounter.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DayCounterImpl](#)  
*abstract base class for day counter implementations*
- class [DayCounter](#)  
*day counter class*

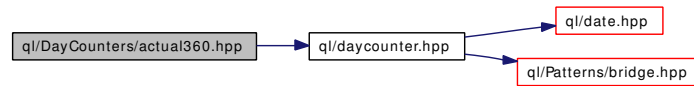
## 8.66 ql/DayCounters/actual360.hpp File Reference

### 8.66.1 Detailed Description

act/360 day counter

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

Include dependency graph for actual360.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Actual360](#)  
*Actual/360 day count convention.*

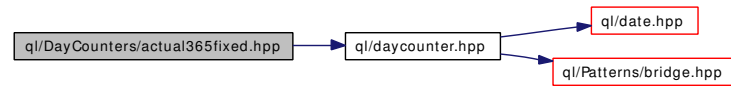
## 8.67 ql/DayCounters/actual365fixed.hpp File Reference

### 8.67.1 Detailed Description

Actual/365 (Fixed) day counter.

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

Include dependency graph for actual365fixed.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Actual365Fixed](#)  
*Actual/365 (Fixed) day count convention.*

## 8.68 ql/DayCounters/actualactual.hpp File Reference

### 8.68.1 Detailed Description

act/act day counters

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

Include dependency graph for actualactual.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ActualActual](#)  
*Actual/Actual day count.*



## 8.69 ql/DayCounters/business252.hpp File Reference

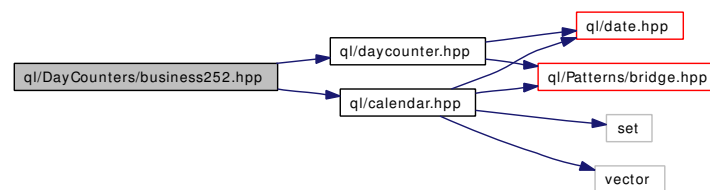
### 8.69.1 Detailed Description

business/252 day counter

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

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

Include dependency graph for business252.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Business252](#)  
*Business/252 day count convention.*

## 8.70 ql/DayCounters/one.hpp File Reference

### 8.70.1 Detailed Description

1/1 day counter

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

Include dependency graph for one.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **OneDayCounter**  
*1/1 day count convention*

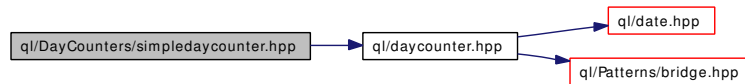
## 8.71 ql/DayCounters/simpliedaycounter.hpp File Reference

### 8.71.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

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

Include dependency graph for `simpliedaycounter.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SimpleDayCounter](#)

*Simple day counter for reproducing theoretical calculations.*

## 8.72 ql/DayCounters/thirty360.hpp File Reference

### 8.72.1 Detailed Description

30/360 day counters

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

Include dependency graph for thirty360.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Thirty360](#)  
*30/360 day count convention*

## 8.73 ql/discretizedasset.hpp File Reference

### 8.73.1 Detailed Description

Discretized asset classes.

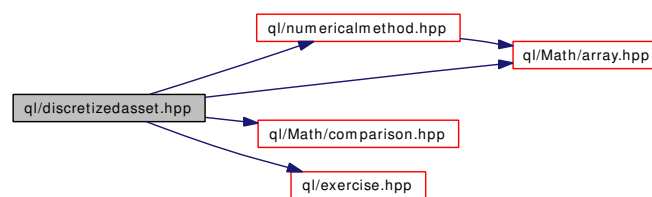
```
#include <ql/numericalmethod.hpp>
```

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Math/comparison.hpp>
```

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

Include dependency graph for discretizedasset.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DiscretizedAsset](#)  
*Discretized asset class used by numerical methods.*
- class [DiscretizedDiscountBond](#)  
*Useful discretized discount bond asset.*
- class [DiscretizedOption](#)  
*Discretized option on a given asset.*

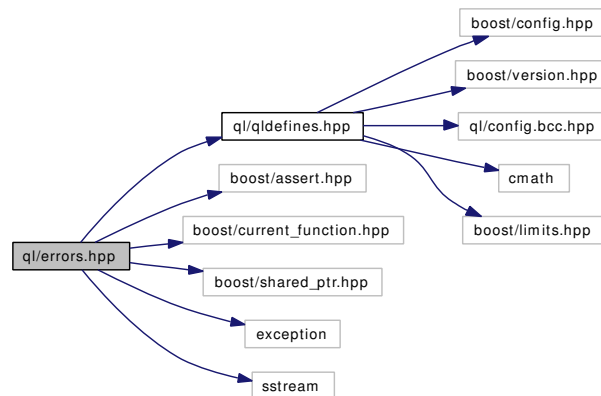
## 8.74 ql/errors.hpp File Reference

### 8.74.1 Detailed Description

Classes and functions for error handling.

```
#include <ql/qldefines.hpp>
#include <boost/assert.hpp>
#include <boost/current_function.hpp>
#include <boost/shared_ptr.hpp>
#include <exception>
#include <sstream>
```

Include dependency graph for errors.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **Error**  
*Base error class.*

## Defines

- #define **QL\_FAIL**(message)  
*throw an error (possibly with file and line information)*
- #define **QL\_ASSERT**(condition, message)  
*throw an error if the given condition is not verified*
- #define **QL\_REQUIRE**(condition, message)

*throw an error if the given pre-condition is not verified*

- `#define QL_ENSURE(condition, message)`  
*throw an error if the given post-condition is not verified*

## 8.74.2 Define Documentation

### 8.74.2.1 `#define QL_FAIL(message)`

**Value:**

```
do { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__, __LINE__, \
                          BOOST_CURRENT_FUNCTION, _ql_msg_stream.str()); \
} while (false)
```

throw an error (possibly with file and line information)

### 8.74.2.2 `#define QL_ASSERT(condition, message)`

**Value:**

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__, __LINE__, \
                          BOOST_CURRENT_FUNCTION, _ql_msg_stream.str()); \
} else
```

throw an error if the given condition is not verified

### 8.74.2.3 `#define QL_REQUIRE(condition, message)`

**Value:**

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__, __LINE__, \
                          BOOST_CURRENT_FUNCTION, _ql_msg_stream.str()); \
} else
```

throw an error if the given pre-condition is not verified

**Examples:**

[DiscreteHedging.cpp](#), and [swapvaluation.cpp](#).

#### 8.74.2.4 #define QL\_ENSURE(condition, message)

**Value:**

```
if (!(condition)) { \
    std::ostringstream _ql_msg_stream; \
    _ql_msg_stream << message; \
    throw QuantLib::Error(__FILE__, __LINE__, \
                          BOOST_CURRENT_FUNCTION, _ql_msg_stream.str()); \
} else
```

throw an error if the given post-condition is not verified



## 8.75 ql/event.hpp File Reference

### 8.75.1 Detailed Description

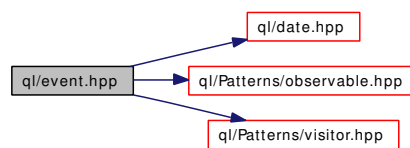
Base class for events associated with a given date.

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

```
#include <ql/Patterns/observable.hpp>
```

```
#include <ql/Patterns/visitor.hpp>
```

Include dependency graph for event.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Event](#)  
*Base class for event.*

## 8.76 ql/exchangerate.hpp File Reference

### 8.76.1 Detailed Description

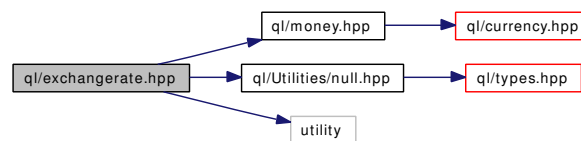
exchange rate between two currencies

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

```
#include <ql/Utilities/null.hpp>
```

```
#include <utility>
```

Include dependency graph for exchangerate.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ExchangeRate](#)  
*exchange rate between two currencies*

## 8.77 ql/exercise.hpp File Reference

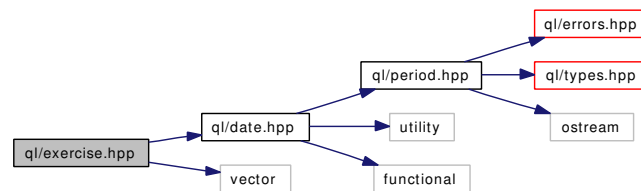
### 8.77.1 Detailed Description

Option exercise classes and payoff function.

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

```
#include <vector>
```

Include dependency graph for exercise.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Exercise](#)  
*Base exercise class.*
- class [EarlyExercise](#)  
*Early-exercise base class.*
- class [AmericanExercise](#)  
*American exercise.*
- class [BermudanExercise](#)  
*Bermudan exercise.*
- class [EuropeanExercise](#)  
*European exercise.*

## 8.78 ql/FiniteDifferences/americancondition.hpp File Reference

### 8.78.1 Detailed Description

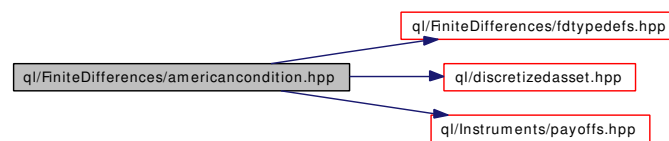
american option exercise condition

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

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

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for americancondition.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AmericanCondition](#)  
*American exercise condition.*

## 8.79 ql/FiniteDifferences/boundarycondition.hpp File Reference

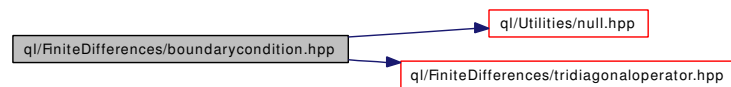
### 8.79.1 Detailed Description

boundary conditions for differential operators

```
#include <ql/Utilities/null.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for boundarycondition.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- 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).*

## 8.80 ql/FiniteDifferences/bsmoperator.hpp File Reference

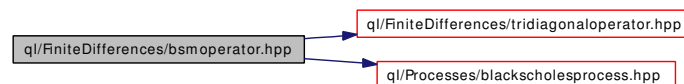
### 8.80.1 Detailed Description

differential operator for Black-Scholes-Merton equation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for bsmoperator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BSMOperator**  
*Black-Scholes-Merton differential operator.*

## 8.81 ql/FiniteDifferences/bsmtermoperator.hpp File Reference

### 8.81.1 Detailed Description

differential operator for Black-Scholes-Merton equation

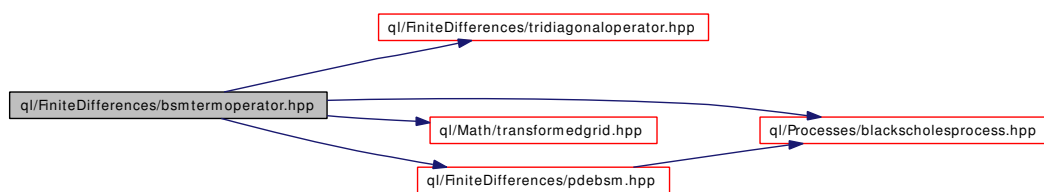
```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/Math/transformedgrid.hpp>
```

```
#include <ql/FiniteDifferences/pdebsm.hpp>
```

Include dependency graph for bsmtermoperator.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `PdeOperator< PdeBSM >` [`BSMTermOperator`](#)  
*Black-Scholes-Merton differential operator.*

## 8.82 ql/FiniteDifferences/cranknicolson.hpp File Reference

### 8.82.1 Detailed Description

Crank-Nicolson scheme for finite difference methods.

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for cranknicolson.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CrankNicolson](#)  
*Crank-Nicolson scheme for finite difference methods.*



## 8.83 ql/FiniteDifferences/dminus.hpp File Reference

### 8.83.1 Detailed Description

*D\_* matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dminus.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DMinus](#)  
*D\_ matricial representation*

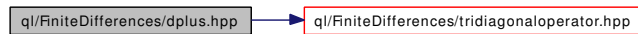
## 8.84 ql/FiniteDifferences/dplus.hpp File Reference

### 8.84.1 Detailed Description

$D_+$  matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dplus.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DPlus](#)  
 *$D_+$  matricial representation*

## 8.85 ql/FiniteDifferences/dplusdminus.hpp File Reference

### 8.85.1 Detailed Description

$D_+D_-$  matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dplusdminus.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **DPlusDMinus**  
 *$D_+D_-$  matricial representation*

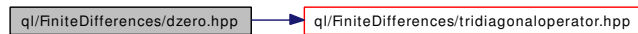
## 8.86 ql/FiniteDifferences/dzero.hpp File Reference

### 8.86.1 Detailed Description

$D_0$  matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dzero.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DZero](#)  
 *$D_0$  matricial representation*

## 8.87 ql/FiniteDifferences/expliciteuler.hpp File Reference

### 8.87.1 Detailed Description

explicit Euler scheme for finite difference methods

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for expliciteuler.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ExplicitEuler](#)  
*Forward Euler scheme for finite difference methods.*

## 8.88 ql/FiniteDifferences/fdtypedefs.hpp File Reference

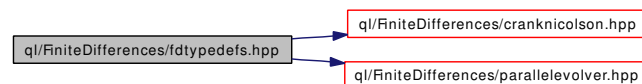
### 8.88.1 Detailed Description

default choices for template instantiations

```
#include <ql/FiniteDifferences/cranknicolson.hpp>
```

```
#include <ql/FiniteDifferences/parallelevolver.hpp>
```

Include dependency graph for fdtypedefs.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `FiniteDifferenceModel< CrankNicolson< TridiagonalOperator > >` [StandardFiniteDifferenceModel](#)  
*default choice for finite-difference model*
- typedef `FiniteDifferenceModel< ParalleEvolver< CrankNicolson< TridiagonalOperator > >` [StandardSystemFiniteDifferenceModel](#)  
*default choice for parallel finite-difference model*
- typedef `StepCondition< Array >` [StandardStepCondition](#)  
*default choice for step condition*
- typedef `CurveDependentStepCondition< Array >` **StandardCurveDependentStepCondition**

## 8.89 ql/FiniteDifferences/finitedifferencemodel.hpp File Reference

### 8.89.1 Detailed Description

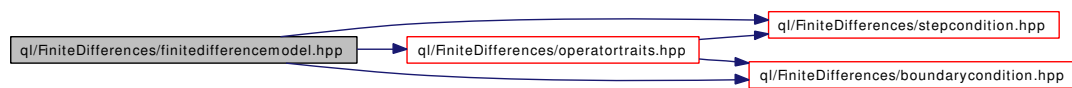
generic finite difference model

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

```
#include <ql/FiniteDifferences/operatortraits.hpp>
```

Include dependency graph for finitedifferencemodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FiniteDifferenceModel](#)  
*Generic finite difference model.*

## 8.90 ql/FiniteDifferences/impliciteuler.hpp File Reference

### 8.90.1 Detailed Description

implicit Euler scheme for finite difference methods

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for impliciteuler.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ImplicitEuler](#)  
*Backward Euler scheme for finite difference methods.*



## 8.91 ql/FiniteDifferences/mixedscheme.hpp File Reference

### 8.91.1 Detailed Description

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

```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
```

Include dependency graph for mixedscheme.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MixedScheme](#)

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

## 8.92 ql/FiniteDifferences/onefactoroperator.hpp File Reference

### 8.92.1 Detailed Description

general differential operator for one-factor interest rate models

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

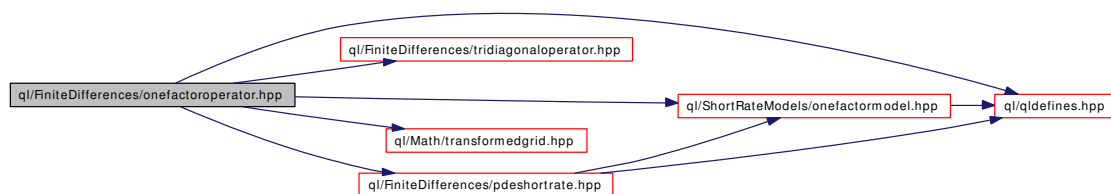
```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

```
#include <ql/Math/transformedgrid.hpp>
```

```
#include <ql/FiniteDifferences/pdeshortrate.hpp>
```

Include dependency graph for onefactoroperator.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `PdeOperator< PdeShortRate >` [OneFactorOperator](#)

*Interest-rate single factor model differential operator.*

## 8.93 ql/FiniteDifferences/operatorfactory.hpp File Reference

### 8.93.1 Detailed Description

Factory for finite difference operators.

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

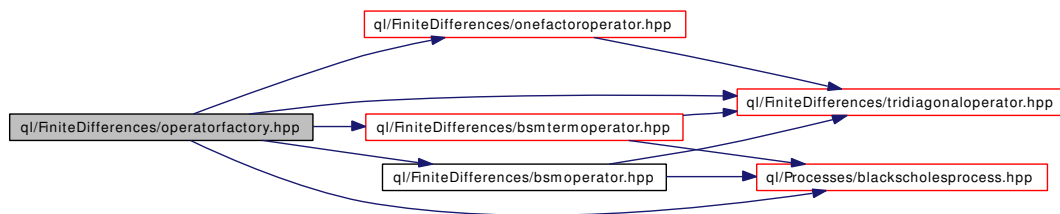
```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/FiniteDifferences/bsmoperator.hpp>
```

```
#include <ql/FiniteDifferences/bsmtermoperator.hpp>
```

```
#include <ql/FiniteDifferences/onefactoroperator.hpp>
```

Include dependency graph for operatorfactory.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [OperatorFactory](#)  
*Black-Scholes-Merton differential operator.*

## 8.94 ql/FiniteDifferences/operatortraits.hpp File Reference

### 8.94.1 Detailed Description

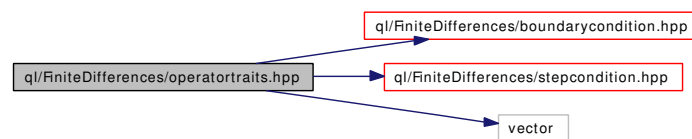
Differential operator traits.

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

```
#include <vector>
```

Include dependency graph for operatortraits.hpp:



### Namespaces

- namespace **QuantLib**

## 8.95 ql/FiniteDifferences/parallelevolver.hpp File Reference

### 8.95.1 Detailed Description

Parallel evolver for multiple arrays.

This class takes the evolver class and creates a new class which evolves each of the evolvers in parallel. Part of what this does is to take the types for each evolver class and then wrapper them so that they create new types which are sets of the old types.

This class is intended to be run in situations where there are parallel differential equations such as with some convertible bond models.

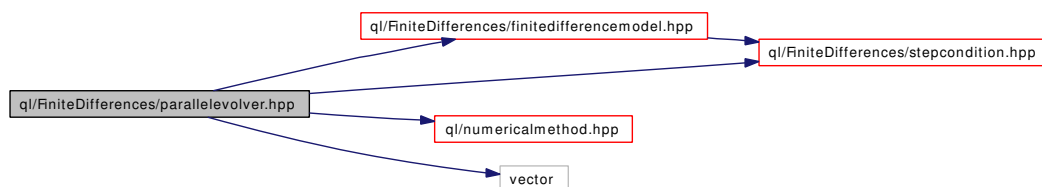
```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
```

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

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

```
#include <vector>
```

Include dependency graph for parallelevolver.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [StepConditionSet](#)  
*Parallel evolver for multiple arrays.*

## 8.96 ql/FiniteDifferences/pde.hpp File Reference

### 8.96.1 Detailed Description

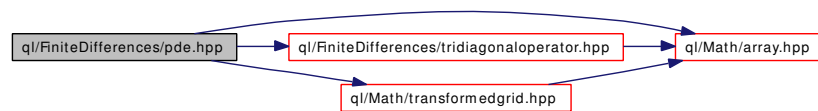
General class for one dimensional PDE's.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/Math/transformedgrid.hpp>
```

Include dependency graph for pde.hpp:



### Namespaces

- namespace **QuantLib**

## 8.97 ql/FiniteDifferences/pdebsm.hpp File Reference

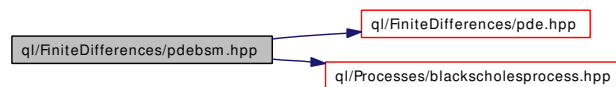
### 8.97.1 Detailed Description

Black-Scholes-Merton PDE.

```
#include <ql/FiniteDifferences/pde.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for pdebsm.hpp:



### Namespaces

- namespace **QuantLib**

## 8.98 ql/FiniteDifferences/pdeshortrate.hpp File Reference

### 8.98.1 Detailed Description

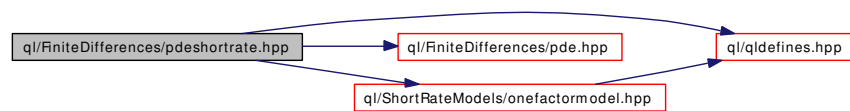
adapter to short rate

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

```
#include <ql/FiniteDifferences/pde.hpp>
```

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for pdeshortrate.hpp:



### Namespaces

- namespace **QuantLib**



## 8.99 ql/FiniteDifferences/shoutcondition.hpp File Reference

### 8.99.1 Detailed Description

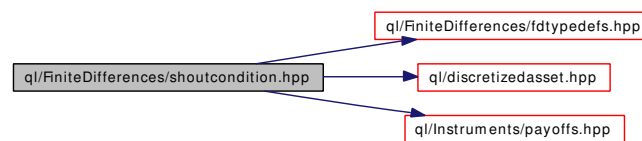
shout option exercise condition

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

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

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for shoutcondition.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ShoutCondition](#)  
*Shout option condition.*

## 8.100 ql/FiniteDifferences/stepcondition.hpp File Reference

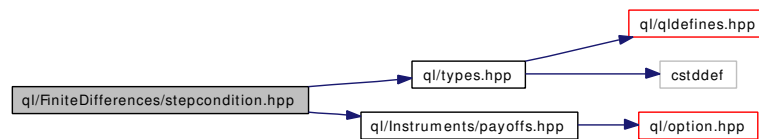
### 8.100.1 Detailed Description

conditions to be applied at every time step

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

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for stepcondition.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [StepCondition](#)  
*condition to be applied at every time step*
- class [NullCondition](#)  
*null step condition*

## 8.101 ql/FiniteDifferences/tridiagonaloperator.hpp File Reference

### 8.101.1 Detailed Description

tridiagonal operator

```
#include <ql/Math/array.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for tridiagonaloperator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TridiagonalOperator](#)  
*Base implementation for tridiagonal operator.*
- class [TridiagonalOperator::TimeSetter](#)  
*encapsulation of time-setting logic*

### Functions

- void **swap** (TridiagonalOperator &, TridiagonalOperator &)
- 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** \* (Real a, const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **operator** \* (const TridiagonalOperator &D, Real a)
- Disposable< TridiagonalOperator > **operator** / (const TridiagonalOperator &D, Real a)

## 8.102 ql/FiniteDifferences/zerocondition.hpp File Reference

### 8.102.1 Detailed Description

zero option exercise condition

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

Include dependency graph for zerocondition.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ZeroCondition](#)  
*Zero exercise condition.*

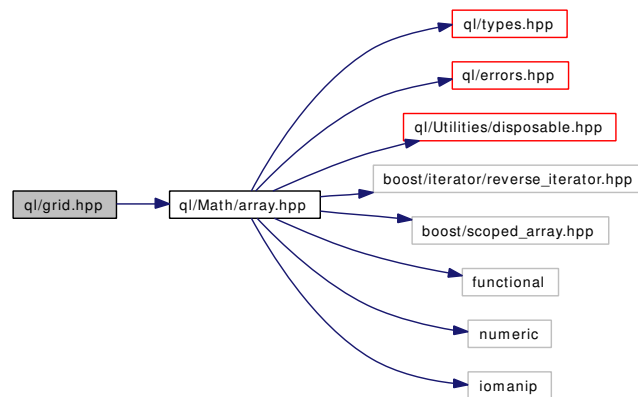
## 8.103 ql/grid.hpp File Reference

### 8.103.1 Detailed Description

Grid constructors.

```
#include <ql/Math/array.hpp>
```

Include dependency graph for grid.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- Disposable< Array > **CenteredGrid** (Real center, Real dx, Size steps)
- Disposable< Array > **BoundedGrid** (Real xMin, Real xMax, Size steps)
- Disposable< Array > **BoundedLogGrid** (Real xMin, Real xMax, Size steps)

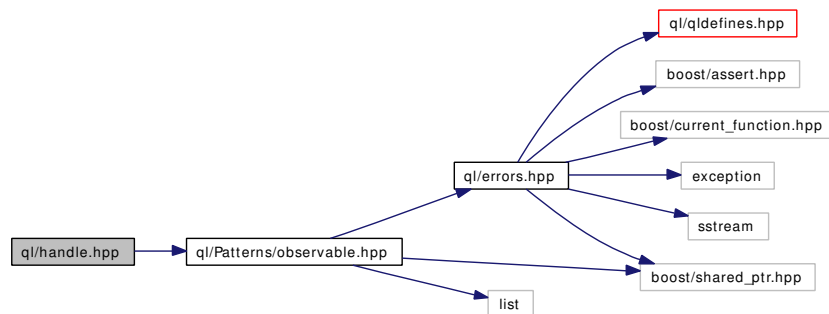
## 8.104 ql/handle.hpp File Reference

### 8.104.1 Detailed Description

Globally accessible relinkable pointer.

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for handle.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Link](#)  
*Relinkable access to a shared pointer.*
- class [Handle](#)  
*Globally accessible relinkable pointer.*

## 8.105 ql/index.hpp File Reference

### 8.105.1 Detailed Description

purely virtual base class for indexes

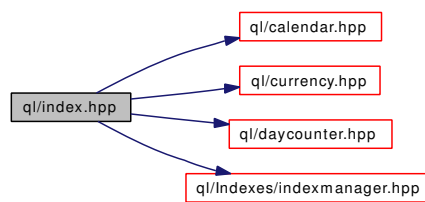
```
#include <ql/calendar.hpp>
```

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

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

```
#include <ql/Indexes/indexmanager.hpp>
```

Include dependency graph for index.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Index](#)  
*purely virtual base class for indexes*

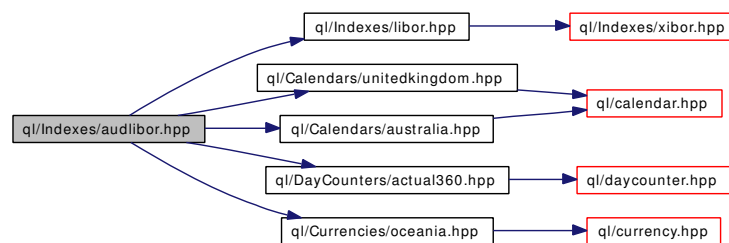
## 8.106 ql/Indexes/audlibor.hpp File Reference

### 8.106.1 Detailed Description

AUD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/australia.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/oceania.hpp>
```

Include dependency graph for audlibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **AUDLibor**  
*AUD LIBOR rate*



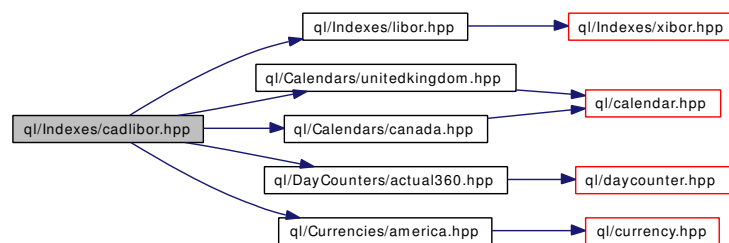
## 8.107 ql/Indexes/cadlibor.hpp File Reference

### 8.107.1 Detailed Description

CAD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/canada.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/america.hpp>
```

Include dependency graph for cadlibor.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **CADLibor**  
*CAD LIBOR rate*

## 8.108 ql/Indexes/cdor.hpp File Reference

### 8.108.1 Detailed Description

CDOR rate

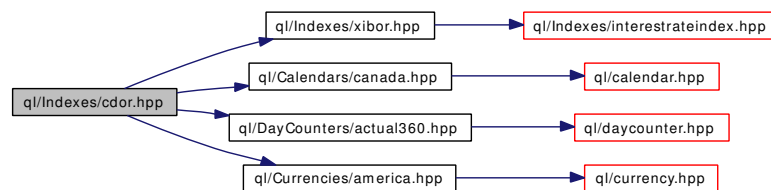
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/canada.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

```
#include <ql/Currencies/america.hpp>
```

Include dependency graph for cdor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Cdor**  
*CDOR rate*

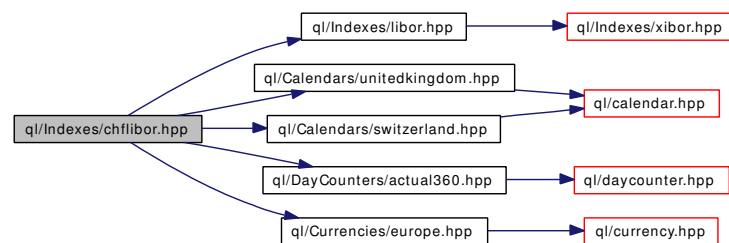
## 8.109 ql/Indexes/chflibor.hpp File Reference

### 8.109.1 Detailed Description

CHF LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/switzerland.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for chflibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **CHFLibor**  
*CHF LIBOR rate*

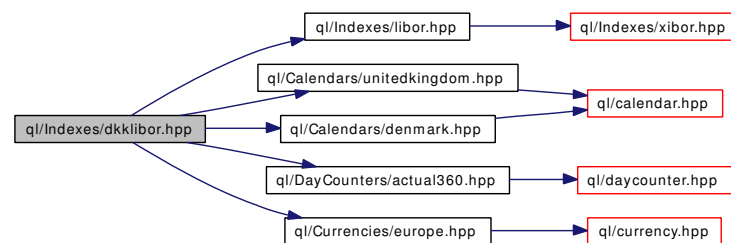
## 8.110 ql/Indexes/dkklibor.hpp File Reference

### 8.110.1 Detailed Description

DKK LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/denmark.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for dkklibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **DKKLibor**  
*DKK LIBOR rate*

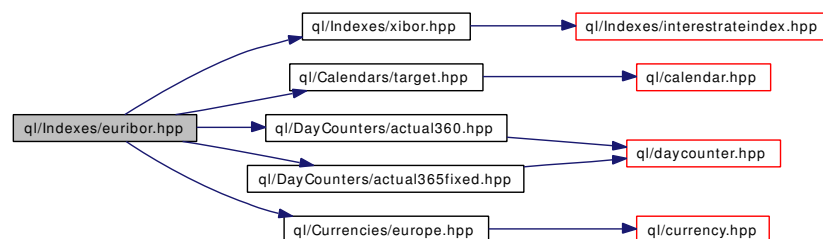
## 8.111 ql/Indexes/euribor.hpp File Reference

### 8.111.1 Detailed Description

Euribor index

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/target.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for euribor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Euribor**  
*Euribor index*
- class **Euribor365**  
*Actual/365 Euribor index.*
- class **EuriborSW**  
*1-week Euribor index*
- class **Euribor2W**  
*2-weeks Euribor index*
- class **Euribor3W**  
*3-weeks Euribor index*
- class **Euribor1M**  
*1-month Euribor index*
- class **Euribor2M**

*2-months Euribor index*

- class [Euribor3M](#)  
*3-months Euribor index*
- class [Euribor4M](#)  
*4-months Euribor index*
- class [Euribor5M](#)  
*5-months Euribor index*
- class [Euribor6M](#)  
*6-months Euribor index*
- class [Euribor7M](#)  
*7-months Euribor index*
- class [Euribor8M](#)  
*8-months Euribor index*
- class [Euribor9M](#)  
*9-months Euribor index*
- class [Euribor10M](#)  
*10-months Euribor index*
- class [Euribor11M](#)  
*11-months Euribor index*
- class [Euribor1Y](#)  
*1-year Euribor index*
- class [Euribor365\\_SW](#)  
*1-week Euribor365 index*
- class [Euribor365\\_2W](#)  
*2-weeks Euribor365 index*
- class [Euribor365\\_3W](#)  
*3-weeks Euribor365 index*
- class [Euribor365\\_1M](#)  
*1-month Euribor365 index*
- class [Euribor365\\_2M](#)  
*2-months Euribor365 index*
- class [Euribor365\\_3M](#)  
*3-months Euribor365 index*

- class [Euribor365\\_4M](#)  
*4-months Euribor365 index*
- class [Euribor365\\_5M](#)  
*5-months Euribor365 index*
- class [Euribor365\\_6M](#)  
*6-months Euribor365 index*
- class [Euribor365\\_7M](#)  
*7-months Euribor365 index*
- class [Euribor365\\_8M](#)  
*8-months Euribor365 index*
- class [Euribor365\\_9M](#)  
*9-months Euribor365 index*
- class [Euribor365\\_10M](#)  
*10-months Euribor365 index*
- class [Euribor365\\_11M](#)  
*11-months Euribor365 index*
- class [Euribor365\\_1Y](#)  
*1-year Euribor365 index*

## 8.112 ql/Indexes/euriborswapfixa.hpp File Reference

### 8.112.1 Detailed Description

euriborswapfixa index

```
#include <ql/Indexes/swapindex.hpp>
```

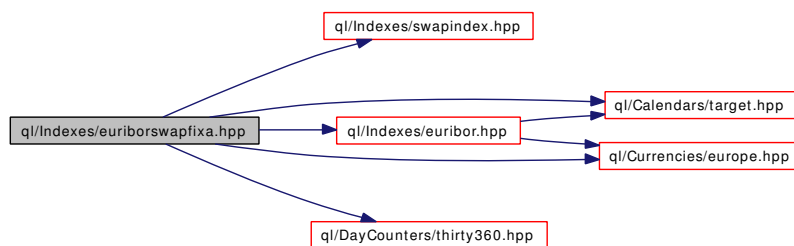
```
#include <ql/Indexes/euribor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/thirty360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for euriborswapfixa.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [EuriborSwapFixA](#)  
*EuriborSwapFixA index*
- class [EuriborSwapFixA1Y](#)  
*1-year EuriborSwapFixA index*
- class [EuriborSwapFixA2Y](#)  
*2-year EuriborSwapFixA index*
- class [EuriborSwapFixA3Y](#)  
*3-year EuriborSwapFixA index*
- class [EuriborSwapFixA4Y](#)  
*4-year EuriborSwapFixA index*
- class [EuriborSwapFixA5Y](#)  
*5-year EuriborSwapFixA index*
- class [EuriborSwapFixA6Y](#)



*6-year EuriborSwapFixA index*

- class [EuriborSwapFixA7Y](#)  
*7-year EuriborSwapFixA index*
- class [EuriborSwapFixA8Y](#)  
*8-year EuriborSwapFixA index*
- class [EuriborSwapFixA9Y](#)  
*9-year EuriborSwapFixA index*
- class [EuriborSwapFixA10Y](#)  
*10-year EuriborSwapFixA index*
- class [EuriborSwapFixA12Y](#)  
*12-year EuriborSwapFixA index*
- class [EuriborSwapFixA15Y](#)  
*15-year EuriborSwapFixA index*
- class [EuriborSwapFixA20Y](#)  
*20-year EuriborSwapFixA index*
- class [EuriborSwapFixA25Y](#)  
*25-year EuriborSwapFixA index*
- class [EuriborSwapFixA30Y](#)  
*30-year EuriborSwapFixA index*

## 8.113 ql/Indexes/euriborswapfixifr.hpp File Reference

### 8.113.1 Detailed Description

EuriborSwapFixIFR index

```
#include <ql/Indexes/swapindex.hpp>
```

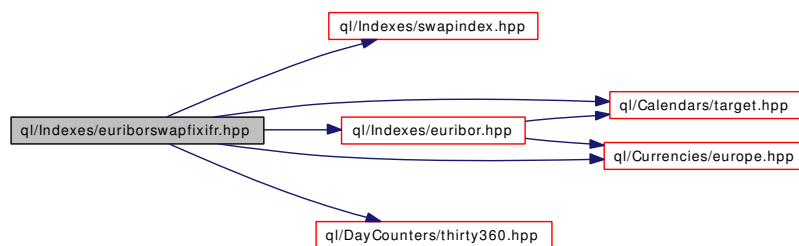
```
#include <ql/Indexes/euribor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/thirty360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for euriborswapfixifr.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [EuriborSwapFixIFR](#)  
*EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR1Y](#)  
*1-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR2Y](#)  
*2-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR3Y](#)  
*3-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR4Y](#)  
*4-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR5Y](#)  
*5-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR6Y](#)

*6-year EuriborSwapFixIFR index*

- class [EuriborSwapFixIFR7Y](#)  
*7-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR8Y](#)  
*8-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR9Y](#)  
*9-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR10Y](#)  
*10-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR12Y](#)  
*12-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR15Y](#)  
*15-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR20Y](#)  
*20-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR25Y](#)  
*25-year EuriborSwapFixIFR index*
- class [EuriborSwapFixIFR30Y](#)  
*30-year EuriborSwapFixIFR index*

## 8.114 ql/Indexes/eurlibor.hpp File Reference

### 8.114.1 Detailed Description

EUR LIBOR rate

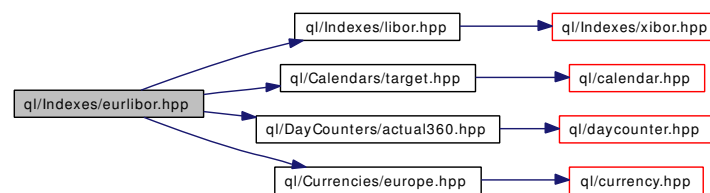
```
#include <ql/Indexes/libor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for eurlibor.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **EURLibor**  
*EUR LIBOR rate*
- class **EURLiborSW**  
*1-week EURLibor index*
- class **EURLibor2W**  
*2-weeks Euribor index*
- class **EURLibor1M**  
*1-month EURLibor index*
- class **EURLibor2M**  
*2-months EURLibor index*
- class **EURLibor3M**  
*3-months EURLibor index*
- class **EURLibor4M**  
*4-months EURLibor index*
- class **EURLibor5M**

*5-months EURLibor index*

- class [EURLibor6M](#)  
*6-months EURLibor index*

- class [EURLibor7M](#)  
*7-months EURLibor index*

- class [EURLibor8M](#)  
*8-months EURLibor index*

- class [EURLibor9M](#)  
*9-months EURLibor index*

- class [EURLibor10M](#)  
*10-months EURLibor index*

- class [EURLibor11M](#)  
*11-months EURLibor index*

- class [EURLibor1Y](#)  
*1-year EURLibor index*

## 8.115 ql/Indexes/eurliborswapfixa.hpp File Reference

### 8.115.1 Detailed Description

eurliborswapfixa index

```
#include <ql/Indexes/swapindex.hpp>
```

```
#include <ql/Indexes/eurlibor.hpp>
```

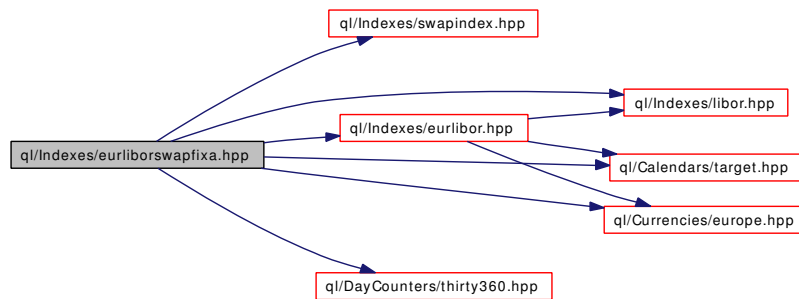
```
#include <ql/Indexes/libor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/thirty360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for eurliborswapfixa.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [EurliborSwapFixA](#)  
*EurliborSwapFixA index*
- class [EurliborSwapFixA1Y](#)  
*1-year EurliborSwapFixA index*
- class [EurliborSwapFixA2Y](#)  
*2-year EurliborSwapFixA index*
- class [EurliborSwapFixA3Y](#)  
*3-year EurliborSwapFixA index*
- class [EurliborSwapFixA4Y](#)  
*4-year EurliborSwapFixA index*
- class [EurliborSwapFixA5Y](#)

*5-year EurliborSwapFixA index*

- class [EurliborSwapFixA6Y](#)  
*6-year EurliborSwapFixA index*
- class [EurliborSwapFixA7Y](#)  
*7-year EurliborSwapFixA index*
- class [EurliborSwapFixA8Y](#)  
*8-year EurliborSwapFixA index*
- class [EurliborSwapFixA9Y](#)  
*9-year EurliborSwapFixA index*
- class [EurliborSwapFixA10Y](#)  
*10-year EurliborSwapFixA index*
- class [EurliborSwapFixA12Y](#)  
*12-year EurliborSwapFixA index*
- class [EurliborSwapFixA15Y](#)  
*15-year EurliborSwapFixA index*
- class [EurliborSwapFixA20Y](#)  
*20-year EurliborSwapFixA index*
- class [EurliborSwapFixA25Y](#)  
*25-year EurliborSwapFixA index*
- class [EurliborSwapFixA30Y](#)  
*30-year EurliborSwapFixA index*

## 8.116 ql/Indexes/eurliborswapfixb.hpp File Reference

### 8.116.1 Detailed Description

eurliborswapfixb index

```
#include <ql/Indexes/swapindex.hpp>
```

```
#include <ql/Indexes/eurlibor.hpp>
```

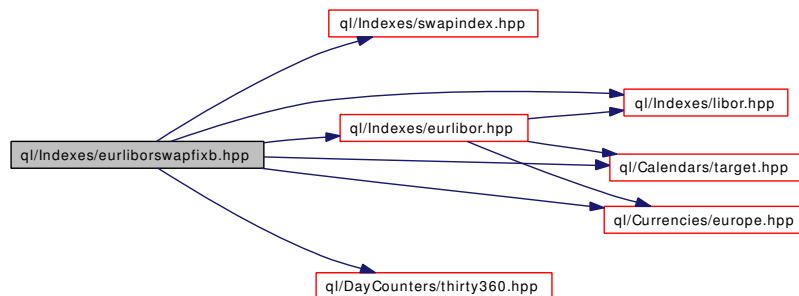
```
#include <ql/Indexes/libor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/thirty360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for eurliborswapfixb.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [EurliborSwapFixB](#)  
*EurliborSwapFixB index*
- class [EurliborSwapFixB1Y](#)  
*1-year EurliborSwapFixB index*
- class [EurliborSwapFixB2Y](#)  
*2-year EurliborSwapFixB index*
- class [EurliborSwapFixB3Y](#)  
*3-year EurliborSwapFixB index*
- class [EurliborSwapFixB4Y](#)  
*4-year EurliborSwapFixB index*
- class [EurliborSwapFixB5Y](#)



*5-year EurliborSwapFixB index*

- class [EurliborSwapFixB6Y](#)  
*6-year EurliborSwapFixB index*
- class [EurliborSwapFixB7Y](#)  
*7-year EurliborSwapFixB index*
- class [EurliborSwapFixB8Y](#)  
*8-year EurliborSwapFixB index*
- class [EurliborSwapFixB9Y](#)  
*9-year EurliborSwapFixB index*
- class [EurliborSwapFixB10Y](#)  
*10-year EurliborSwapFixB index*
- class [EurliborSwapFixB12Y](#)  
*12-year EurliborSwapFixB index*
- class [EurliborSwapFixB15Y](#)  
*15-year EurliborSwapFixB index*
- class [EurliborSwapFixB20Y](#)  
*20-year EurliborSwapFixB index*
- class [EurliborSwapFixB25Y](#)  
*25-year EurliborSwapFixB index*
- class [EurliborSwapFixB30Y](#)  
*30-year EurliborSwapFixB index*

## 8.117 ql/Indexes/eurliborswapfixifr.hpp File Reference

### 8.117.1 Detailed Description

eurliborswapfixifr index

```
#include <ql/Indexes/swapindex.hpp>
```

```
#include <ql/Indexes/eurlibor.hpp>
```

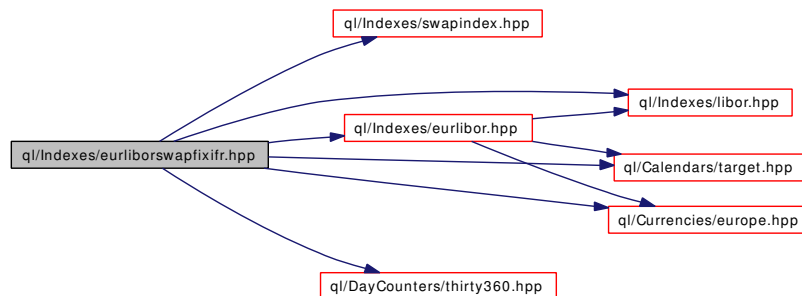
```
#include <ql/Indexes/libor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/thirty360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for eurliborswapfixifr.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [EurliborSwapFixIFR](#)  
*EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR1Y](#)  
*1-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR2Y](#)  
*2-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR3Y](#)  
*3-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR4Y](#)  
*4-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR5Y](#)

*5-year EurliborSwapFixIFR index*

- class [EurliborSwapFixIFR6Y](#)  
*6-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR7Y](#)  
*7-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR8Y](#)  
*8-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR9Y](#)  
*9-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR10Y](#)  
*10-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR12Y](#)  
*12-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR15Y](#)  
*15-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR20Y](#)  
*20-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR25Y](#)  
*25-year EurliborSwapFixIFR index*
- class [EurliborSwapFixIFR30Y](#)  
*30-year EurliborSwapFixIFR index*

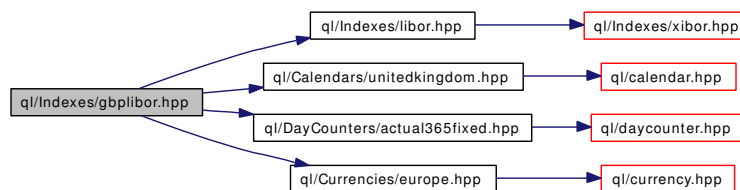
## 8.118 ql/Indexes/gbplibor.hpp File Reference

### 8.118.1 Detailed Description

GBP LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for gbplibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **GBPLibor**  
*GBP LIBOR rate*

## 8.119 ql/Indexes/indexmanager.hpp File Reference

### 8.119.1 Detailed Description

global repository for past index fixings

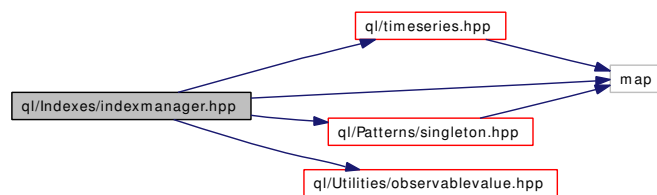
```
#include <ql/timeseries.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <ql/Utilities/observablevalue.hpp>
```

```
#include <map>
```

Include dependency graph for indexmanager.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [IndexManager](#)  
*global repository for past index fixings*

## 8.120 ql/Indexes/interestrateindex.hpp File Reference

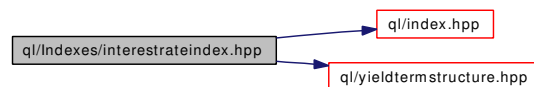
### 8.120.1 Detailed Description

base class for interest rate indexes

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

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

Include dependency graph for interestrateindex.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterestRateIndex](#)  
*base class for interest rate indexes*

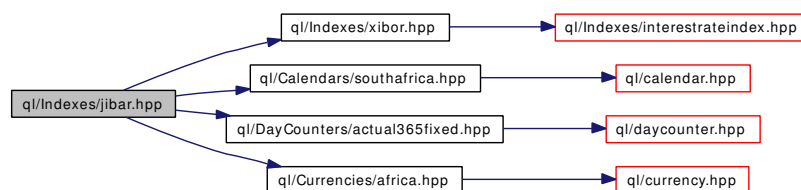
## 8.121 ql/Indexes/jibar.hpp File Reference

### 8.121.1 Detailed Description

JIBAR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/southafrica.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
#include <ql/Currencies/africa.hpp>
```

Include dependency graph for jibar.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Jibar**  
*JIBAR rate*

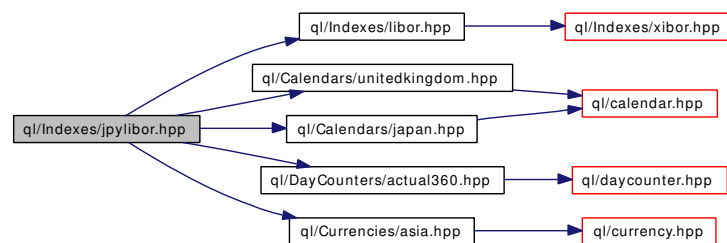
## 8.122 ql/Indexes/jpylibor.hpp File Reference

### 8.122.1 Detailed Description

JPY LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/japan.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/asia.hpp>
```

Include dependency graph for jpylibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **JPYLibor**  
*JPY LIBOR rate*



## 8.123 ql/Indexes/libor.hpp File Reference

### 8.123.1 Detailed Description

base class for BBA LIBOR indexes

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for libor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Libor**  
*base class for BBA LIBOR indexes*

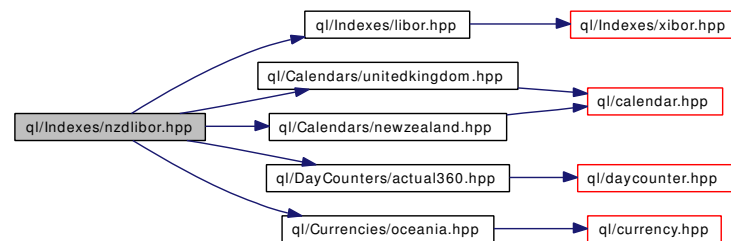
## 8.124 ql/Indexes/nzdlibor.hpp File Reference

### 8.124.1 Detailed Description

NZD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/newzealand.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/oceania.hpp>
```

Include dependency graph for nzdlibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **NZDLibor**  
*NZD LIBOR rate*

## 8.125 ql/Indexes/swapindex.hpp File Reference

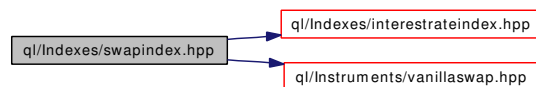
### 8.125.1 Detailed Description

swap-rate indexes

```
#include <ql/Indexes/interestrateindex.hpp>
```

```
#include <ql/Instruments/vanillaswap.hpp>
```

Include dependency graph for swapindex.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwapIndex](#)  
*base class for swap-rate indexes*

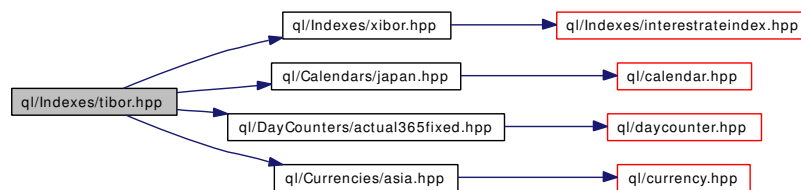
## 8.126 ql/Indexes/tibor.hpp File Reference

### 8.126.1 Detailed Description

JPY TIBOR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/japan.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
#include <ql/Currencies/asia.hpp>
```

Include dependency graph for tibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Tibor**  
*JPY TIBOR index*

## 8.127 ql/Indexes/trlibor.hpp File Reference

### 8.127.1 Detailed Description

TRY LIBOR rate

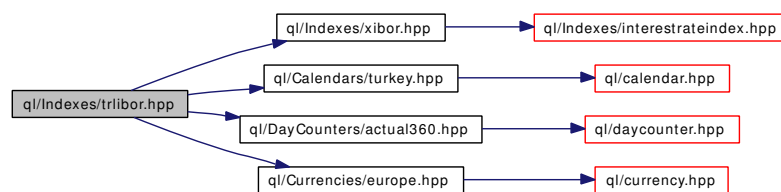
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/turkey.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for trlibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **TRLibor**  
*TRY LIBOR rate*

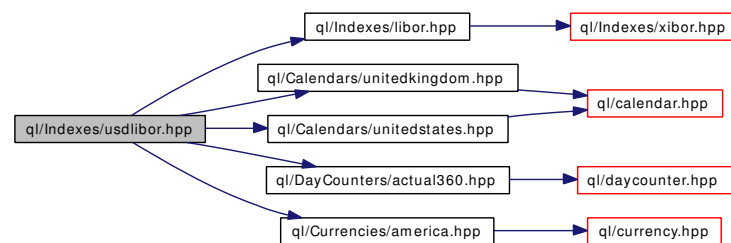
## 8.128 ql/Indexes/usdlibor.hpp File Reference

### 8.128.1 Detailed Description

USD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/unitedstates.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/america.hpp>
```

Include dependency graph for usdlibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **USDLibor**  
*USD LIBOR rate*

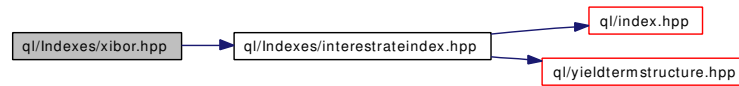
## 8.129 ql/Indexes/xibor.hpp File Reference

### 8.129.1 Detailed Description

base class for LIBOR-like indexes

```
#include <ql/Indexes/interestrateindex.hpp>
```

Include dependency graph for xibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Xibor**  
*base class for LIBOR-like indexes*

## 8.130 ql/Indexes/zibor.hpp File Reference

### 8.130.1 Detailed Description

CHF ZIBOR rate

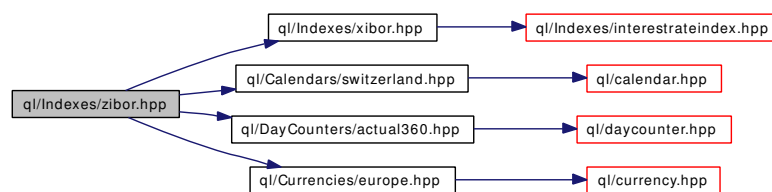
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/switzerland.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for zibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Zibor](#)  
*CHF ZIBOR rate*



## 8.131 ql/instrument.hpp File Reference

### 8.131.1 Detailed Description

Abstract instrument class.

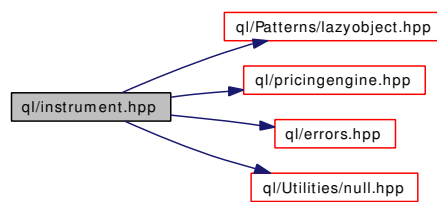
```
#include <ql/Patterns/lazyobject.hpp>
```

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

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

```
#include <ql/Utilities/null.hpp>
```

Include dependency graph for instrument.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Instrument**  
*Abstract instrument class.*
- class **Value**  
*pricing results*

## 8.132 ql/Instruments/asianoption.hpp File Reference

### 8.132.1 Detailed Description

Asian option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for asianoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct [Average](#)  
*placeholder for enumerated averaging types*
- class [ContinuousAveragingAsianOption](#)  
*Continuous-averaging Asian option.*
- class [DiscreteAveragingAsianOption](#)  
*Discrete-averaging Asian option.*
- class [DiscreteAveragingAsianOption::arguments](#)  
*Extra arguments for single-asset discrete-average Asian option.*
- class [ContinuousAveragingAsianOption::arguments](#)  
*Extra arguments for single-asset continuous-average Asian option.*
- class [DiscreteAveragingAsianOption::engine](#)  
*Discrete-averaging Asian engine base class.*
- class [ContinuousAveragingAsianOption::engine](#)  
*Continuous-averaging Asian engine base class.*

## 8.133 ql/Instruments/assetswap.hpp File Reference

### 8.133.1 Detailed Description

Bullet Bond vs Libor swap.

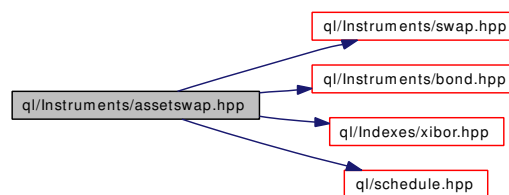
```
#include <ql/Instruments/swap.hpp>
```

```
#include <ql/Instruments/bond.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

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

Include dependency graph for assetswap.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AssetSwap](#)  
*Asset swap.*
- class [AssetSwap::arguments](#)  
*Arguments for asset swap calculation*
- class [AssetSwap::results](#)  
*Results from simple swap calculation*

## 8.134 ql/Instruments/barrieroption.hpp File Reference

### 8.134.1 Detailed Description

Barrier option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for barrieroption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct **Barrier**  
*Placeholder for enumerated barrier types.*
- class **BarrierOption**  
*Barrier option on a single asset.*
- class **BarrierOption::arguments**  
*Arguments for barrier option calculation*
- class **BarrierOption::engine**  
*Barrier engine base class*

## 8.135 ql/Instruments/basketoption.hpp File Reference

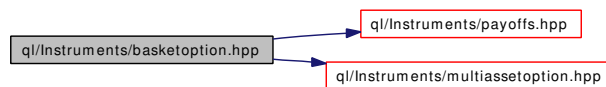
### 8.135.1 Detailed Description

Basket option on a number of assets.

```
#include <ql/Instruments/payoffs.hpp>
```

```
#include <ql/Instruments/multiassetoption.hpp>
```

Include dependency graph for basketoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BasketOption**  
*Basket option on a number of assets.*
- class **BasketOption::arguments**  
*Arguments for basket option calculation*
- class **BasketOption::engine**  
*Basket option engine base class*

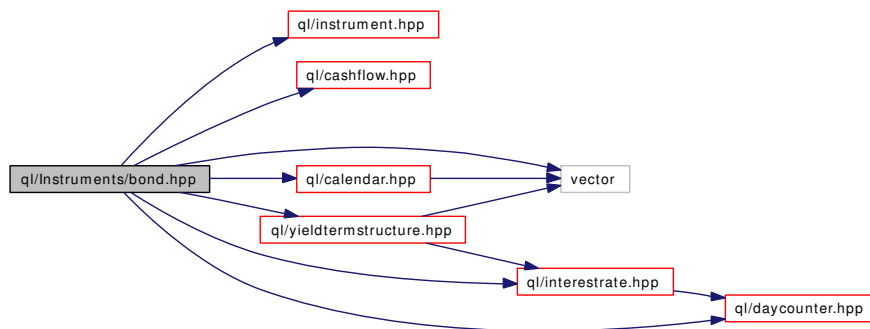
## 8.136 ql/Instruments/bond.hpp File Reference

### 8.136.1 Detailed Description

concrete bond class

```
#include <ql/instrument.hpp>
#include <ql/cashflow.hpp>
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/interestrate.hpp>
#include <ql/yieldtermstructure.hpp>
#include <vector>
```

Include dependency graph for bond.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Bond**  
*Base bond class.*

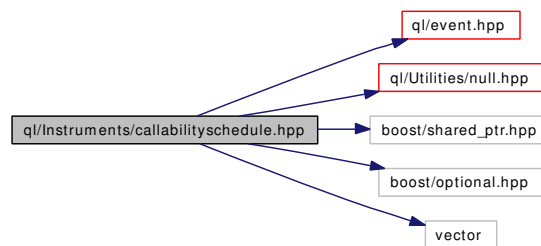
## 8.137 ql/Instruments/callabilityschedule.hpp File Reference

### 8.137.1 Detailed Description

Schedule of put/call dates.

```
#include <ql/event.hpp>
#include <ql/Utilities/null.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/optional.hpp>
#include <vector>
```

Include dependency graph for callabilityschedule.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Callability](#)  
*instrument callability*
- class [Callability::Price](#)  
*amount to be paid upon callability*

### Typedefs

- typedef `std::vector< boost::shared_ptr< Callability > >` **CallabilitySchedule**

## 8.138 ql/Instruments/capfloor.hpp File Reference

### 8.138.1 Detailed Description

Cap and Floor class.

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

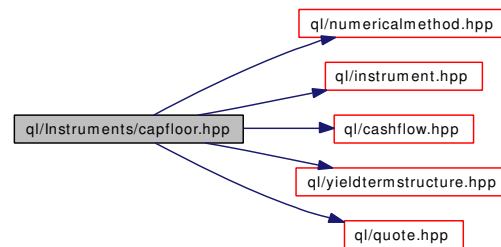
```
#include <ql/instrument.hpp>
```

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

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

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

Include dependency graph for capfloor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CapFloor](#)  
*Base class for cap-like instruments.*
- class [Cap](#)  
*Concrete cap class.*
- class [Floor](#)  
*Concrete floor class.*
- class [Collar](#)  
*Concrete collar class.*
- class [CapFloor::arguments](#)  
*Arguments for cap/floor calculation*
- class [CapFloor::results](#)  
*Results from cap/floor calculation*
- class [CapFloor::engine](#)



*base class for cap/floor engines*

## Functions

- `std::ostream & operator<< (std::ostream &, CapFloor::Type)`

## 8.139 ql/Instruments/cliquestoption.hpp File Reference

### 8.139.1 Detailed Description

Cliquet option.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for cliquestoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CliquetOption](#)  
*cliquet (Ratchet) option*
- class [CliquetOption::arguments](#)  
*Arguments for cliquet option calculation*
- class [CliquetOption::engine](#)  
*Cliquet engine base class.*

## 8.140 ql/Instruments/compositeinstrument.hpp File Reference

### 8.140.1 Detailed Description

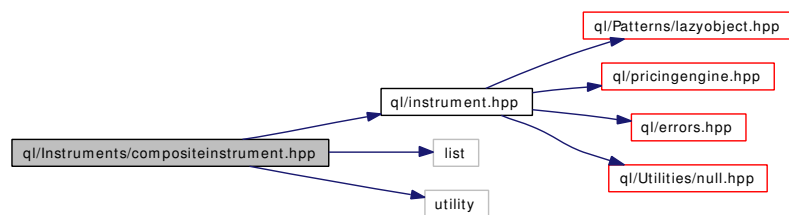
Composite instrument class.

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

```
#include <list>
```

```
#include <utility>
```

Include dependency graph for compositeinstrument.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **CompositeInstrument**  
*Composite instrument.*

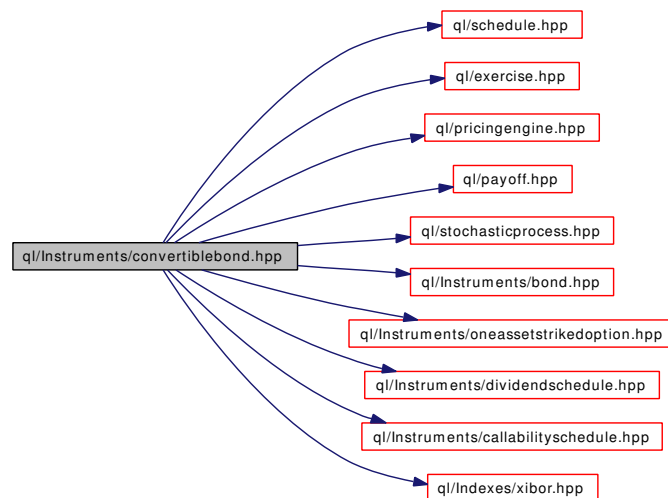
## 8.141 ql/Instruments/convertiblebond.hpp File Reference

### 8.141.1 Detailed Description

convertible bond class

```
#include <ql/schedule.hpp>
#include <ql/exercise.hpp>
#include <ql/pricingengine.hpp>
#include <ql/payoff.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/Instruments/bond.hpp>
#include <ql/Instruments/oneassetstrikedoption.hpp>
#include <ql/Instruments/dividendschedule.hpp>
#include <ql/Instruments/callabilityschedule.hpp>
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for convertiblebond.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **SoftCallability**  
*callability leaving to the holder the possibility to convert*
- class **ConvertibleZeroCouponBond**  
*convertible zero-coupon bond*

- class [ConvertibleFixedCouponBond](#)  
*convertible fixed-coupon bond*
- class [ConvertibleFloatingRateBond](#)  
*convertible floating-rate bond*
- class [ConvertibleBond::option::arguments](#)  
*Arguments for Convertible [Bond](#) calculation*
- class [ConvertibleBond::option::engine](#)  
*convertible bond engine base class*

## 8.142 ql/Instruments/dividendschedule.hpp File Reference

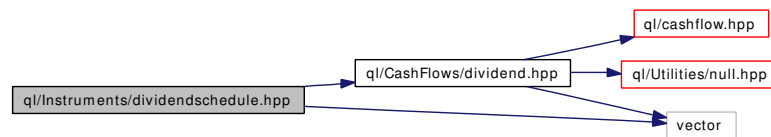
### 8.142.1 Detailed Description

Schedule of dividend dates.

```
#include <ql/CashFlows/dividend.hpp>
```

```
#include <vector>
```

Include dependency graph for dividendschedule.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `std::vector< boost::shared_ptr< Dividend > >` **DividendSchedule**

## 8.143 ql/Instruments/dividendvanillaoption.hpp File Reference

### 8.143.1 Detailed Description

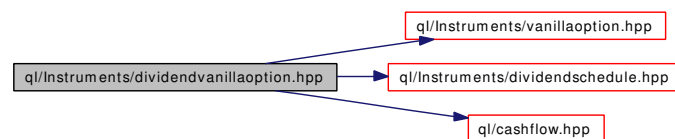
Vanilla option on a single asset with discrete dividends.

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/Instruments/dividendschedule.hpp>
```

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

Include dependency graph for dividendvanillaoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DividendVanillaOption](#)  
*Single-asset vanilla option (no barriers) with discrete dividends.*
- class [DividendVanillaOption::arguments](#)  
*Arguments for dividend vanilla option calculation*
- class [DividendVanillaOption::engine](#)  
*Dividend vanilla option engine base class.*

## 8.144 ql/Instruments/europeanoption.hpp File Reference

### 8.144.1 Detailed Description

European option on a single asset.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for europeanoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [EuropeanOption](#)  
*European option on a single asset.*



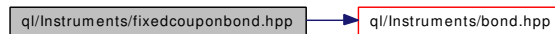
## 8.145 ql/Instruments/fixedcouponbond.hpp File Reference

### 8.145.1 Detailed Description

fixed-coupon bond

```
#include <ql/Instruments/bond.hpp>
```

Include dependency graph for fixedcouponbond.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **FixedCouponBond**  
*fixed-coupon bond*

## 8.146 ql/Instruments/fixedcouponbondforward.hpp File Reference

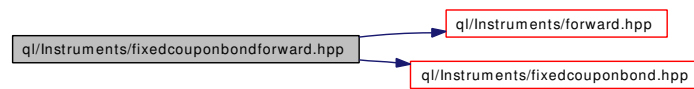
### 8.146.1 Detailed Description

forward contract on a fixed-coupon bond

```
#include <ql/Instruments/forward.hpp>
```

```
#include <ql/Instruments/fixedcouponbond.hpp>
```

Include dependency graph for fixedcouponbondforward.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **FixedCouponBondForward**  
*forward contract on a fixed-coupon bond*

## 8.147 ql/Instruments/floatingratebond.hpp File Reference

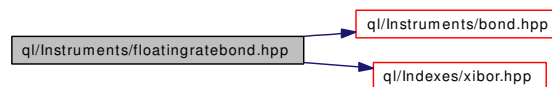
### 8.147.1 Detailed Description

floating-rate bond

```
#include <ql/Instruments/bond.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for floatingratebond.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **FloatingRateBond**  
*floating-rate bond*

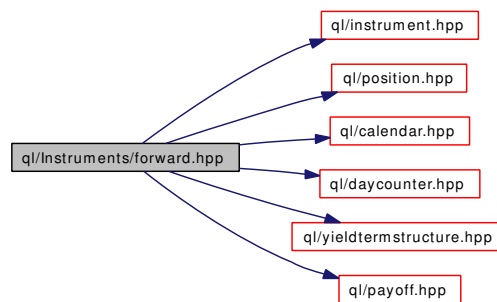
## 8.148 ql/Instruments/forward.hpp File Reference

### 8.148.1 Detailed Description

Base forward class.

```
#include <ql/instrument.hpp>
#include <ql/position.hpp>
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/payoff.hpp>
```

Include dependency graph for forward.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Forward](#)  
*Abstract base forward class.*
- class [ForwardTypePayoff](#)  
*Class for forward type payoffs.*

## 8.149 ql/Instruments/forwardrateagreement.hpp File Reference

### 8.149.1 Detailed Description

forward rate agreement

```
#include <ql/Instruments/forward.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for forwardrateagreement.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ForwardRateAgreement](#)  
*Forward* rate agreement (FRA) class.

## 8.150 ql/Instruments/forwardvanillaoption.hpp File Reference

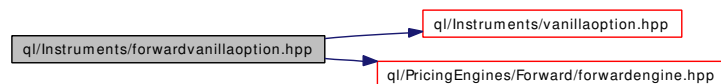
### 8.150.1 Detailed Description

Forward version of a vanilla option.

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

Include dependency graph for forwardvanillaoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ForwardVanillaOption](#)  
*Forward version of a vanilla option.*

## 8.151 ql/Instruments/lookbackoption.hpp File Reference

### 8.151.1 Detailed Description

Lookback option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for lookbackoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ContinuousFloatingLookbackOption](#)  
*Continuous-floating lookback option.*
- class [ContinuousFixedLookbackOption](#)  
*Continuous-fixed lookback option.*
- class [ContinuousFloatingLookbackOption::arguments](#)  
*Arguments for continuous floating lookback option calculation*
- class [ContinuousFixedLookbackOption::arguments](#)  
*Arguments for continuous fixed lookback option calculation*
- class [ContinuousFloatingLookbackOption::engine](#)  
*Continuous floating lookback engine base class*
- class [ContinuousFixedLookbackOption::engine](#)  
*Continuous fixed lookback engine base class*

## 8.152 ql/Instruments/multiassetoption.hpp File Reference

### 8.152.1 Detailed Description

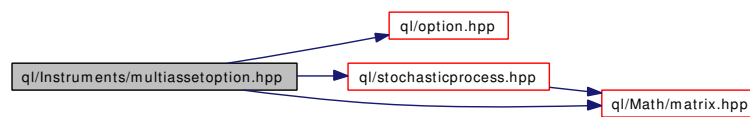
Option on multiple assets.

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

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

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for multiassetoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MultiAssetOption](#)  
*Base class for options on multiple assets.*
- class [MultiAssetOption::arguments](#)  
*Arguments for multi-asset option calculation*
- class [MultiAssetOption::results](#)  
*Results from multi-asset option calculation*



## 8.153 ql/Instruments/oneassetoption.hpp File Reference

### 8.153.1 Detailed Description

Option on a single asset.

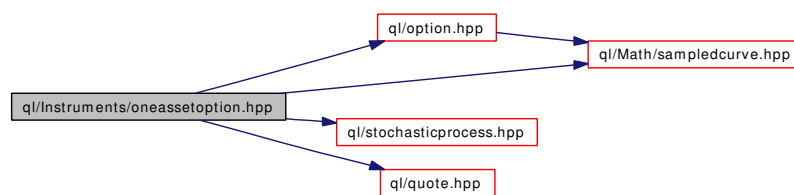
```
#include <ql/option.hpp>
```

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

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

```
#include <ql/Math/sampledcurve.hpp>
```

Include dependency graph for oneassetoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [OneAssetOption](#)  
*Base class for options on a single asset.*
- class [OneAssetOption::arguments](#)  
*Arguments for single-asset option calculation*
- class [OneAssetOption::results](#)  
*Results from single-asset option calculation*

## 8.154 ql/Instruments/oneassetstrikedoption.hpp File Reference

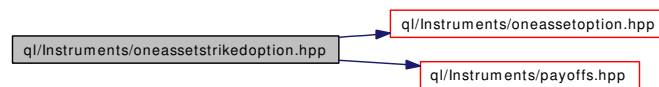
### 8.154.1 Detailed Description

Option on a single asset with striked payoff.

```
#include <ql/Instruments/oneassetoption.hpp>
```

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for oneassetstrikedoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [OneAssetStrikedOption](#)

*Base class for options on a single asset with striked payoff.*

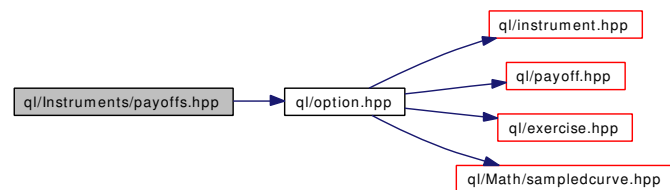
## 8.155 ql/Instruments/payoffs.hpp File Reference

### 8.155.1 Detailed Description

Payoffs for various options.

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

Include dependency graph for `payoffs.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TypePayoff](#)  
*Intermediate class for call/put/straddle payoffs.*
- class [StrikedTypePayoff](#)  
*Intermediate class for payoffs based on a fixed strike.*
- class [FloatingTypePayoff](#)  
*Payoff based on a floating strike.*
- class [PlainVanillaPayoff](#)  
*Plain-vanilla payoff.*
- class [PercentageStrikePayoff](#)  
*Payoff with strike expressed as percentage*
- class [CashOrNothingPayoff](#)  
*Binary cash-or-nothing payoff.*
- class [AssetOrNothingPayoff](#)  
*Binary asset-or-nothing payoff.*
- class [GapPayoff](#)  
*Binary gap payoff.*
- class [SuperSharePayoff](#)  
*Binary supershare payoff.*

## 8.156 ql/Instruments/quantoforwardvanillaoption.hpp File Reference

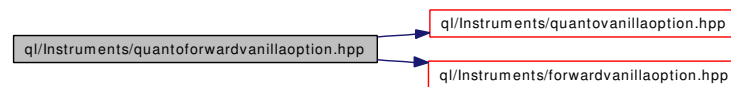
### 8.156.1 Detailed Description

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**

### Classes

- class [QuantoForwardVanillaOption](#)  
*Quanto version of a forward vanilla option.*

## 8.157 ql/Instruments/quantovanillaoption.hpp File Reference

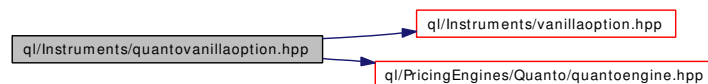
### 8.157.1 Detailed Description

Quanto version of a vanilla option.

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

Include dependency graph for quantovanillaoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [QuantoVanillaOption](#)  
*quanto version of a vanilla option*

## 8.158 ql/Instruments/stock.hpp File Reference

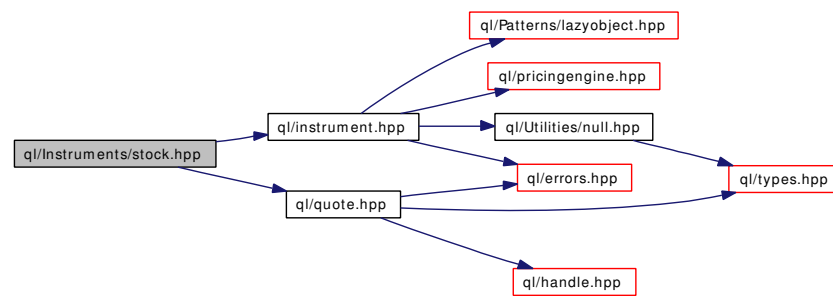
### 8.158.1 Detailed Description

concrete stock class

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

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

Include dependency graph for stock.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Stock](#)  
*Simple stock class.*

## 8.159 ql/Instruments/swap.hpp File Reference

### 8.159.1 Detailed Description

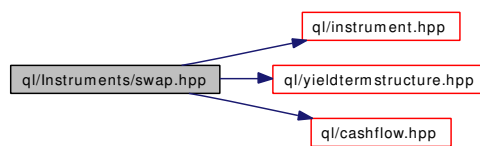
Interest rate swap.

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

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

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

Include dependency graph for swap.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Swap](#)  
*Interest rate swap.*

## 8.160 ql/Instruments/swaption.hpp File Reference

### 8.160.1 Detailed Description

Swaption class.

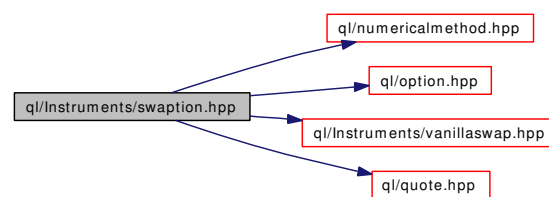
```
#include <ql/numericalmethod.hpp>
```

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

```
#include <ql/Instruments/vanillaswap.hpp>
```

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

Include dependency graph for swaption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct **Settlement**  
*settlement information*
- class **Swaption**  
*Swaption class*
- class **Swaption::arguments**  
*Arguments for swaption calculation*
- class **Swaption::results**  
*Results from swaption calculation*
- class **Swaption::engine**  
*base class for swaption engines*



## 8.161 ql/Instruments/vanillaoption.hpp File Reference

### 8.161.1 Detailed Description

Vanilla option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for vanillaoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [VanillaOption](#)  
*Vanilla option (no discrete dividends, no barriers) on a single asset.*
- class [VanillaOption::engine](#)  
*Vanilla option engine base class.*

## 8.162 ql/Instruments/vanillaswap.hpp File Reference

### 8.162.1 Detailed Description

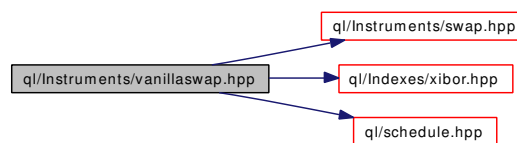
Simple fixed-rate vs Libor swap.

```
#include <ql/Instruments/swap.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

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

Include dependency graph for `vanillaswap.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class **VanillaSwap**  
*Plain-vanilla swap.*
- class **VanillaSwap::arguments**  
*Arguments for simple swap calculation*
- class **VanillaSwap::results**  
*Results from simple swap calculation*
- class **MakeVanillaSwap**  
*helper class*

## 8.163 ql/Instruments/varianceswap.hpp File Reference

### 8.163.1 Detailed Description

Variance Swap.

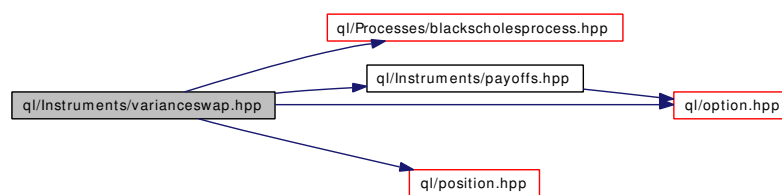
```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/Instruments/payoffs.hpp>
```

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

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

Include dependency graph for varianceswap.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **VarianceSwap**  
*Variance swap.*
- class **VarianceSwap::arguments**  
*Arguments for forward fair-variance calculation*
- class **VarianceSwap::results**  
*Results from variance-swap calculation*
- class **VarianceSwap::engine**  
*base class for variance-swap engines*

## 8.164 ql/Instruments/zerocouponbond.hpp File Reference

### 8.164.1 Detailed Description

zero-coupon bond

```
#include <ql/Instruments/bond.hpp>
```

Include dependency graph for zerocouponbond.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ZeroCouponBond](#)  
*zero-coupon bond*

## 8.165 ql/interestrate.hpp File Reference

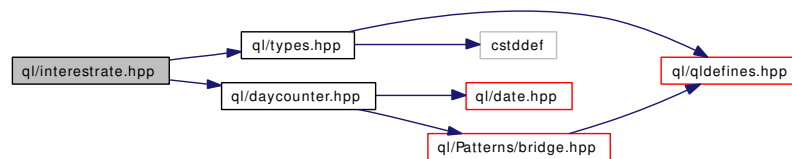
### 8.165.1 Detailed Description

Instrument rate class.

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

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

Include dependency graph for interestrate.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterestRate](#)  
*Concrete interest rate class.*

### Enumerations

- enum **Compounding** { [Simple](#) = 0, [Compounded](#) = 1, [Continuous](#) = 2, [SimpleThenCompounded](#) }  
*Interest rate compounding rule.*

## 8.166 ql/Lattices/binomialestree.hpp File Reference

### 8.166.1 Detailed Description

Binomial tree class.

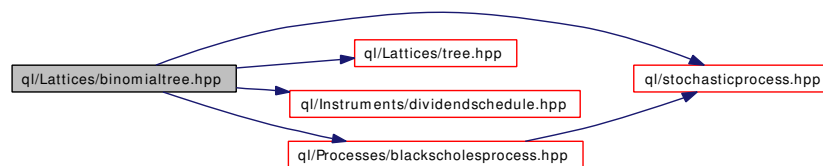
```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/Lattices/tree.hpp>
```

```
#include <ql/Instruments/dividendschedule.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for binomialestree.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **BinomialTree**  
*Binomial tree base class.*
- class **EqualProbabilitiesBinomialTree**  
*Base class for equal probabilities binomial tree.*
- class **EqualJumpsBinomialTree**  
*Base class for equal jumps binomial tree.*
- class **JarrowRudd**  
*Jarrow-Rudd (multiplicative) equal probabilities binomial tree.*
- class **CoxRossRubinstein**  
*Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.*
- class **AdditiveEQPBinomialTree**  
*Additive equal probabilities binomial tree.*
- class **Trigeorgis**  
*Trigeorgis (additive equal jumps) binomial tree*
- class **Tian**

*Tian tree: third moment matching, multiplicative approach*

- class [LeisenReimer](#)

*Leisen & Reimer tree: multiplicative approach.*

## 8.167 ql/Lattices/bsmlattice.hpp File Reference

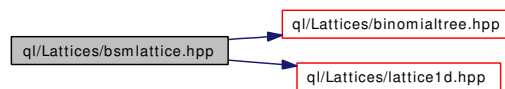
### 8.167.1 Detailed Description

Binomial trees under the BSM model.

```
#include <ql/Lattices/binomialtree.hpp>
```

```
#include <ql/Lattices/lattice1d.hpp>
```

Include dependency graph for bsmlattice.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BlackScholesLattice](#)

*Simple binomial lattice approximating the Black-Scholes model.*



## 8.168 ql/Lattices/lattice.hpp File Reference

### 8.168.1 Detailed Description

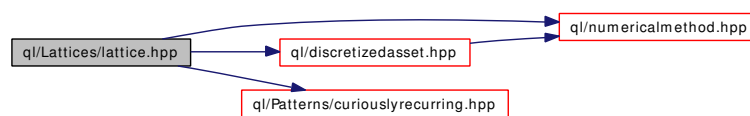
Lattice method class.

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

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

```
#include <ql/Patterns/curiouslyrecurring.hpp>
```

Include dependency graph for lattice.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Lattice](#)  
*Lattice-method base class.*

## 8.169 ql/Lattices/lattice1d.hpp File Reference

### 8.169.1 Detailed Description

One-dimensional lattice class.

```
#include <ql/Lattices/lattice.hpp>
```

Include dependency graph for lattice1d.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Lattice1D](#)  
*One-dimensional lattice.*

## 8.170 ql/Lattices/lattice2d.hpp File Reference

### 8.170.1 Detailed Description

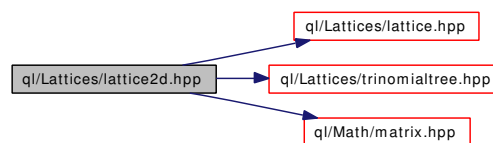
Two-dimensional lattice class.

```
#include <ql/Lattices/lattice.hpp>
```

```
#include <ql/Lattices/trinomialtree.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for lattice2d.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Lattice2D](#)  
*Two-dimensional lattice.*

## 8.171 ql/Lattices/tflattice.hpp File Reference

### 8.171.1 Detailed Description

Binomial Tsiveriotis-Fernandes tree model.

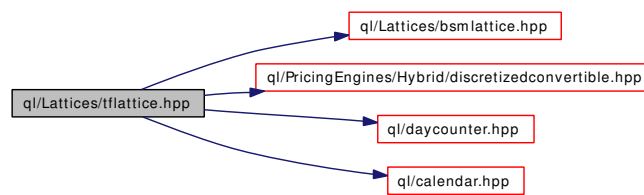
```
#include <ql/Lattices/bsmlattice.hpp>
```

```
#include <ql/PricingEngines/Hybrid/discretizedconvertible.hpp>
```

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

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

Include dependency graph for tflattice.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [TsiveriotisFernandesLattice](#)

*Binomial lattice approximating the Tsiveriotis-Fernandes model.*

## 8.172 ql/Lattices/tree.hpp File Reference

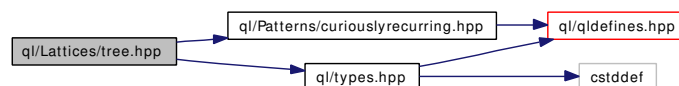
### 8.172.1 Detailed Description

Tree class.

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

```
#include <ql/Patterns/curiouslyrecurring.hpp>
```

Include dependency graph for tree.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Tree](#)

*Tree approximating a single-factor diffusion*

## 8.173 ql/Lattices/trinomialtree.hpp File Reference

### 8.173.1 Detailed Description

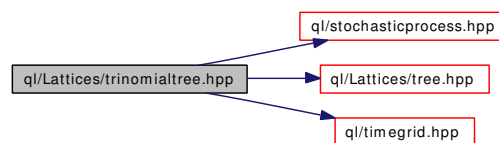
Trinomial tree class.

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

```
#include <ql/Lattices/tree.hpp>
```

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

Include dependency graph for trinomialtree.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TrinomialTree](#)  
*Recombining trinomial tree class.*

## 8.174 ql/MarketModels/driftdrcalculator.hpp File Reference

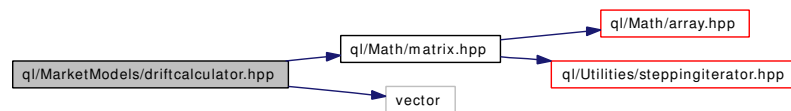
### 8.174.1 Detailed Description

Drift computation for Market Model.

```
#include <ql/Math/matrix.hpp>
```

```
#include <vector>
```

Include dependency graph for driftdrcalculator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DriftCalculator](#)  
*Drift computation for Market Models.*

## 8.175 ql/Math/array.hpp File Reference

### 8.175.1 Detailed Description

1-D array used in linear algebra.

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

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

```
#include <ql/Utilities/disposable.hpp>
```

```
#include <boost/iterator/reverse_iterator.hpp>
```

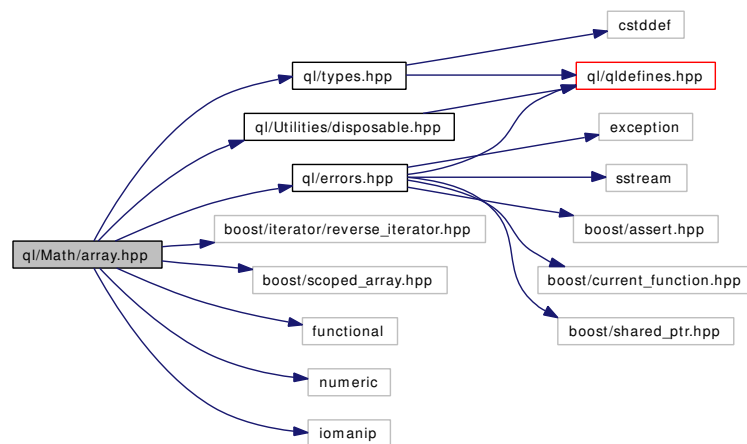
```
#include <boost/scoped_array.hpp>
```

```
#include <functional>
```

```
#include <numeric>
```

```
#include <iomanip>
```

Include dependency graph for array.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Array](#)  
*1-D array used in linear algebra.*



## 8.176 ql/Math/backwardflatinterpolation.hpp File Reference

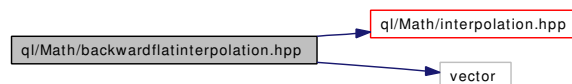
### 8.176.1 Detailed Description

backward-flat interpolation between discrete points

```
#include <ql/Math/interpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for backwardflatinterpolation.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class [BackwardFlatInterpolation](#)  
*Backward-flat interpolation between discrete points.*
- class [BackwardFlat](#)  
*Backward-flat interpolation factory and traits.*

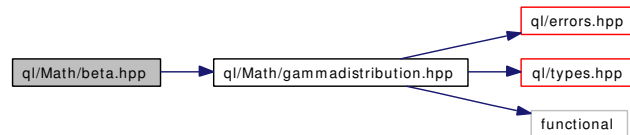
## 8.177 ql/Math/beta.hpp File Reference

### 8.177.1 Detailed Description

Beta and beta incomplete functions.

```
#include <ql/Math/gammadistribution.hpp>
```

Include dependency graph for beta.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- Real **betaFunction** (Real *z*, Real *w*)
- Real **betaContinuedFraction** (Real *a*, Real *b*, Real *x*, Real *accuracy*=1e-16, Integer *max-Iteration*=100)
- Real **incompleteBetaFunction** (Real *a*, Real *b*, Real *x*, Real *accuracy*=1e-16, Integer *max-Iteration*=100)

*Incomplete Beta function.*

## 8.178 ql/Math/bicubicsplineinterpolation.hpp File Reference

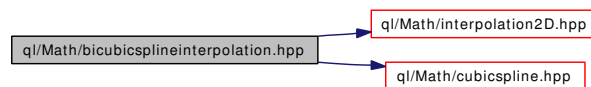
### 8.178.1 Detailed Description

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::detail**

### Classes

- class [BicubicSpline](#)
- class [Bicubic](#)  
*bicubic-spline interpolation factory*

## 8.179 ql/Math/bilinearinterpolation.hpp File Reference

### 8.179.1 Detailed Description

bilinear interpolation between discrete points

```
#include <ql/Math/interpolation2D.hpp>
```

Include dependency graph for bilinearinterpolation.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class **BilinearInterpolation**  
*bilinear interpolation between discrete points*
- class **Bilinear**  
*bilinear interpolation factory*

## 8.180 ql/Math/binomialdistribution.hpp File Reference

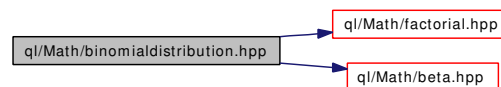
### 8.180.1 Detailed Description

Binomial distribution.

```
#include <ql/Math/factorial.hpp>
```

```
#include <ql/Math/beta.hpp>
```

Include dependency graph for binomialdistribution.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BinomialDistribution](#)  
*Binomial probability distribution function.*
- class [CumulativeBinomialDistribution](#)  
*Cumulative binomial distribution function.*

### Functions

- Real **binomialCoefficientLn** (BigNatural n, BigNatural k)
- Real **binomialCoefficient** (BigNatural n, BigNatural k)
- Real [PeizerPrattMethod2Inversion](#) (Real z, BigNatural n)

## 8.181 ql/Math/bivariatenormaldistribution.hpp File Reference

### 8.181.1 Detailed Description

bivariate cumulative normal distribution

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for bivariatenormaldistribution.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BivariateCumulativeNormalDistributionDr78](#)  
*Cumulative bivariate normal distribution function.*
- class [BivariateCumulativeNormalDistributionWe04DP](#)  
*Cumulative bivariate normal distribution function (West 2004).*

### Typedefs

- typedef `BivariateCumulativeNormalDistributionWe04DP` [BivariateCumulativeNormalDistribution](#)  
*default bivariate implementation*

## 8.182 ql/Math/chisquairedistribution.hpp File Reference

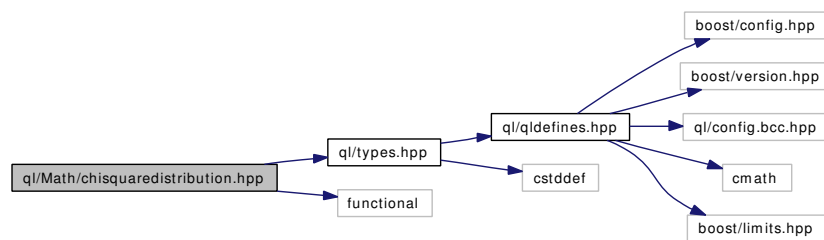
### 8.182.1 Detailed Description

Chi-square (central and non-central) distributions.

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

```
#include <functional>
```

Include dependency graph for chisquairedistribution.hpp:



### Namespaces

- namespace **QuantLib**

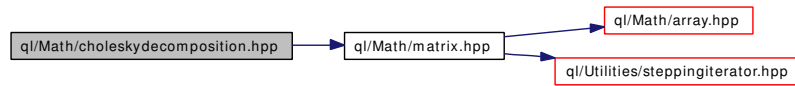
## 8.183 ql/Math/choleskydecomposition.hpp File Reference

### 8.183.1 Detailed Description

Cholesky decomposition.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for choleskydecomposition.hpp:



### Namespaces

- namespace **QuantLib**



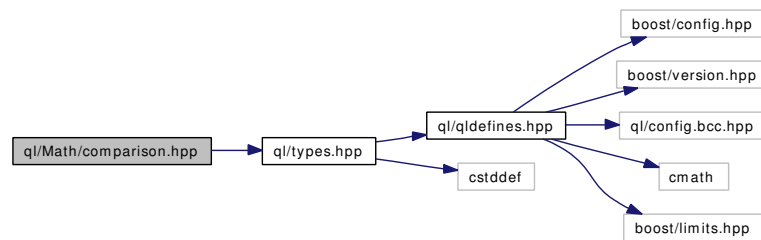
## 8.184 ql/Math/comparison.hpp File Reference

### 8.184.1 Detailed Description

floating-point comparisons

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

Include dependency graph for comparison.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- bool `close` (Real x, Real y)
- bool `close` (Real x, Real y, Size n)
- bool `close_enough` (Real x, Real y)
- bool `close_enough` (Real x, Real y, Size n)

## 8.185 ql/Math/convergencestatistics.hpp File Reference

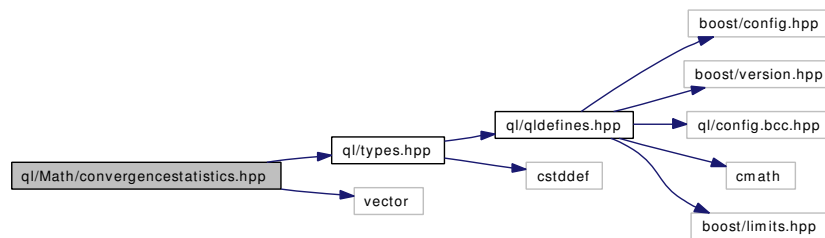
### 8.185.1 Detailed Description

statistics tool with risk measures

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

```
#include <vector>
```

Include dependency graph for convergencestatistics.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ConvergenceStatistics](#)  
*statistics class with convergence table*

## 8.186 ql/Math/cubicspline.hpp File Reference

### 8.186.1 Detailed Description

cubic spline interpolation between discrete points

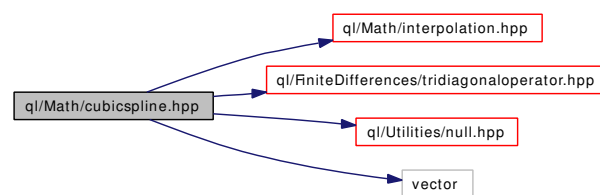
```
#include <ql/Math/interpolation.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

```
#include <vector>
```

Include dependency graph for cubicspline.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class [CubicSpline](#)  
*Cubic spline interpolation between discrete points.*
- class [MonotonicCubicSpline](#)  
*Cubic spline with monotonicity constraint*
- class [NaturalCubicSpline](#)  
*Cubic spline with null second derivative at end points*
- class [NaturalMonotonicCubicSpline](#)  
*Natural cubic spline with monotonicity constraint.*
- class [Cubic](#)  
*cubic-spline interpolation factory and traits*

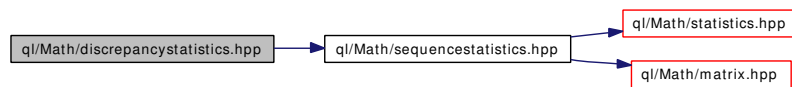
## 8.187 ql/Math/discrepancystatistics.hpp File Reference

### 8.187.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

```
#include <ql/Math/sequencestatistics.hpp>
```

Include dependency graph for `discrepancystatistics.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DiscrepancyStatistics](#)  
*Statistic tool for sequences with discrepancy calculation.*

## 8.188 ql/Math/errorfunction.hpp File Reference

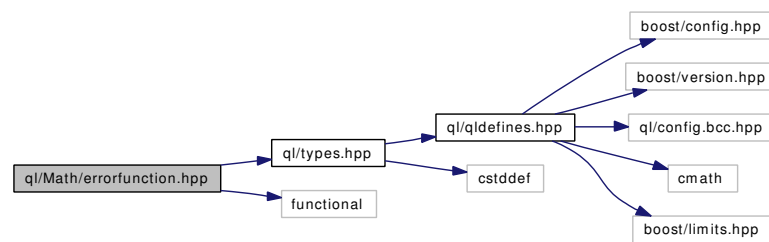
### 8.188.1 Detailed Description

Error function.

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

```
#include <functional>
```

Include dependency graph for errorfunction.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **ErrorFunction**  
*Error function*

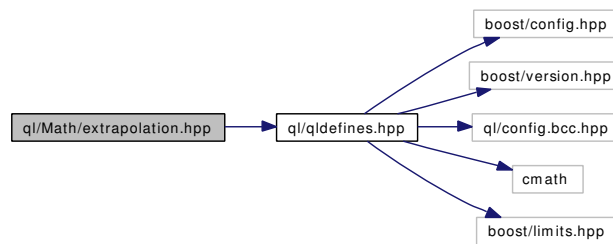
## 8.189 ql/Math/extrapolation.hpp File Reference

### 8.189.1 Detailed Description

class-wide extrapolation settings

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

Include dependency graph for extrapolation.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Extrapolator](#)  
*base class for classes possibly allowing extrapolation*

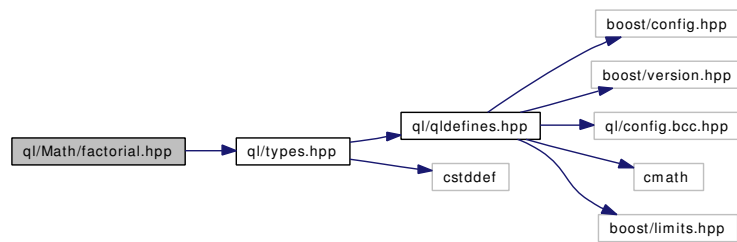
## 8.190 ql/Math/factorial.hpp File Reference

### 8.190.1 Detailed Description

Factorial numbers calculator.

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

Include dependency graph for factorial.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Factorial](#)  
*Factorial numbers calculator*

## 8.191 ql/Math/forwardflatinterpolation.hpp File Reference

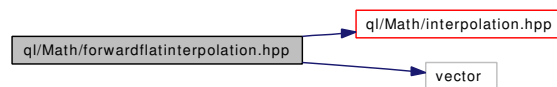
### 8.191.1 Detailed Description

forward-flat interpolation between discrete points

```
#include <ql/Math/interpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for forwardflatinterpolation.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class [ForwardFlatInterpolation](#)  
*Forward-flat interpolation between discrete points.*
- class [ForwardFlat](#)  
*Forward-flat interpolation factory and traits.*



## 8.192 ql/Math/functional.hpp File Reference

### 8.192.1 Detailed Description

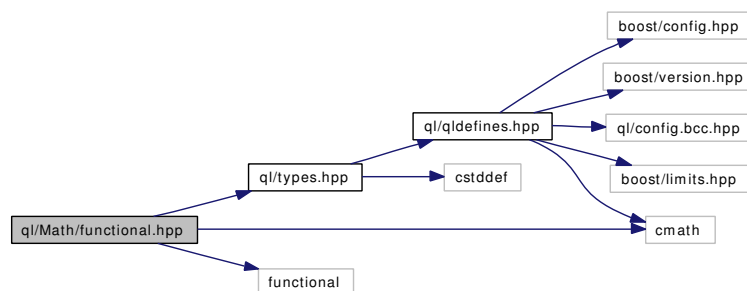
functionals and combinators not included in the STL

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

```
#include <cmath>
```

```
#include <functional>
```

Include dependency graph for functional.hpp:



### Namespaces

- namespace **QuantLib**

### 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)`
- `template<class F, class G, class H> binary_compose3_function< F, G, H > compose3 (const F &f, const G &g, const H &h)`

## 8.193 ql/Math/gammadistribution.hpp File Reference

### 8.193.1 Detailed Description

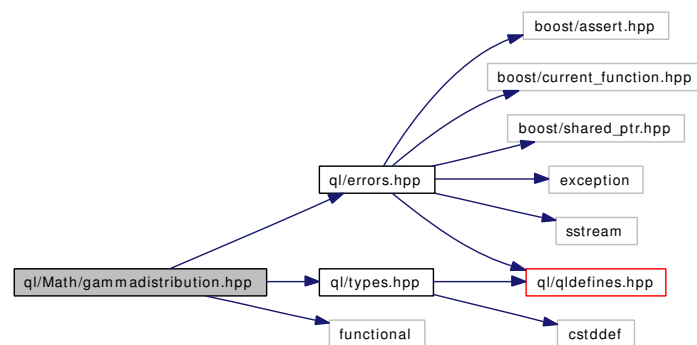
Gamma distribution.

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

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

```
#include <functional>
```

Include dependency graph for gammadistribution.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GammaFunction](#)  
*Gamma function class.*

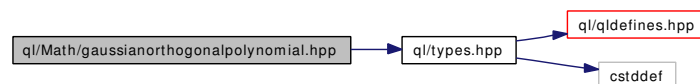
## 8.194 ql/Math/gaussianorthogonalpolynomial.hpp File Reference

### 8.194.1 Detailed Description

orthogonal polynomials for gaussian quadratures

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

Include dependency graph for gaussianorthogonalpolynomial.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GaussianOrthogonalPolynomial](#)  
*orthogonal polynomial for Gaussian quadratures*
- class [GaussLaguerrePolynomial](#)  
*Gauss-Laguerre polynomial.*
- class [GaussHermitePolynomial](#)  
*Gauss-Hermite polynomial.*
- class [GaussJacobiPolynomial](#)  
*Gauss-Jacobi polynomial.*
- class [GaussHyperbolicPolynomial](#)  
*Gauss hyperbolic polynomial.*

## 8.195 ql/Math/gaussianquadratures.hpp File Reference

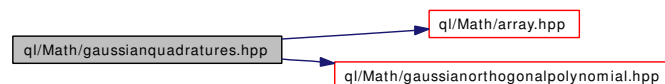
### 8.195.1 Detailed Description

Integral of a 1-dimensional function using the Gauss quadratures.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Math/gaussianorthogonalpolynomial.hpp>
```

Include dependency graph for gaussianquadratures.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GaussianQuadrature](#)  
*Integral of a 1-dimensional function using the Gauss quadratures method.*
- class [GaussLaguerreIntegration](#)  
*generalized Gauss-Laguerre integration*
- class [GaussHermiteIntegration](#)  
*generalized Gauss-Hermite integration*
- class [GaussJacobiIntegration](#)  
*Gauss-Jacobi integration.*
- class [GaussHyperbolicIntegration](#)  
*Gauss-Hyperbolic integration.*
- class [GaussLegendreIntegration](#)  
*Gauss-Legendre integration.*
- class [GaussChebyshevIntegration](#)  
*Gauss-Chebyshev integration.*
- class [GaussChebyshev2thIntegration](#)  
*Gauss-Chebyshev integration second kind.*
- class [GaussGegenbauerIntegration](#)  
*Gauss-Gegenbauer integration.*
- class [TabulatedGaussLegendre](#)

*tabulated Gauss-Legendre quadratures*

## 8.196 ql/Math/gaussianstatistics.hpp File Reference

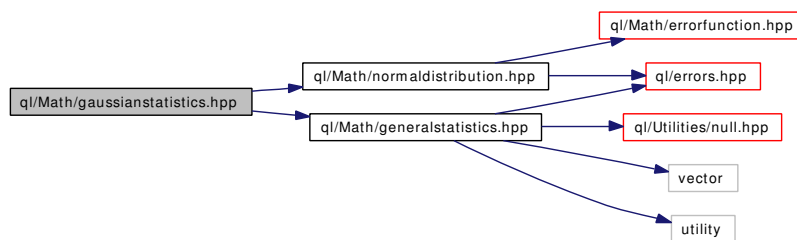
### 8.196.1 Detailed Description

statistics tool for gaussian-assumption risk measures

```
#include <ql/Math/normaldistribution.hpp>
```

```
#include <ql/Math/generalstatistics.hpp>
```

Include dependency graph for gaussianstatistics.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GenericGaussianStatistics](#)  
*Statistics tool for gaussian-assumption risk measures.*
- class [StatsHolder](#)  
*Helper class for precomputed distributions.*

### Typedefs

- typedef `GenericGaussianStatistics< GeneralStatistics >` [GaussianStatistics](#)  
*default gaussian statistic tool*

## 8.197 ql/Math/generalstatistics.hpp File Reference

### 8.197.1 Detailed Description

statistics tool

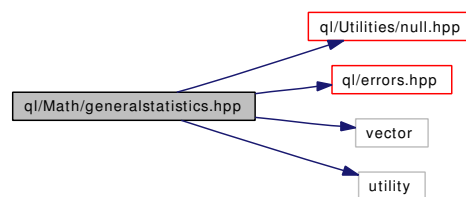
```
#include <ql/Utilities/null.hpp>
```

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

```
#include <vector>
```

```
#include <utility>
```

Include dependency graph for generalstatistics.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GeneralStatistics](#)  
*Statistics tool.*

## 8.198 ql/Math/incompletegamma.hpp File Reference

### 8.198.1 Detailed Description

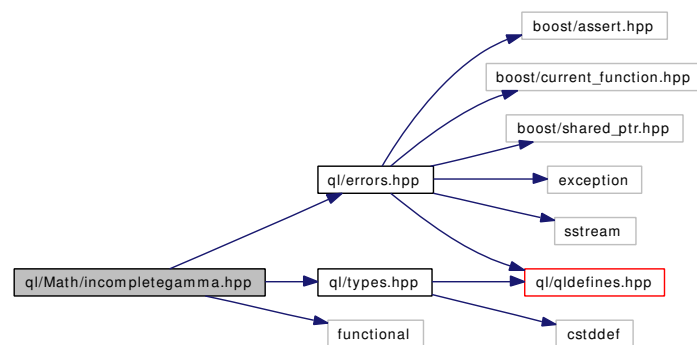
Incomplete Gamma function.

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

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

```
#include <functional>
```

Include dependency graph for incompletegamma.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- Real [incompleteGammaFunction](#) (Real a, Real x, Real accuracy=1.0e-13, Integer maxIteration=100)  
*Incomplete Gamma function.*
- Real **incompleteGammaFunctionSeriesRepr** (Real a, Real x, Real accuracy=1.0e-13, Integer maxIteration=100)
- Real **incompleteGammaFunctionContinuedFractionRepr** (Real a, Real x, Real accuracy=1.0e-13, Integer maxIteration=100)



## 8.199 ql/Math/incrementalstatistics.hpp File Reference

### 8.199.1 Detailed Description

statistics tool based on incremental accumulation

```
#include <ql/Utilities/null.hpp>
```

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

Include dependency graph for incrementalstatistics.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [IncrementalStatistics](#)  
*Statistics tool based on incremental accumulation.*

## 8.200 ql/Math/interpolation.hpp File Reference

### 8.200.1 Detailed Description

base class for 1-D interpolations

```
#include <ql/Patterns/bridge.hpp>
```

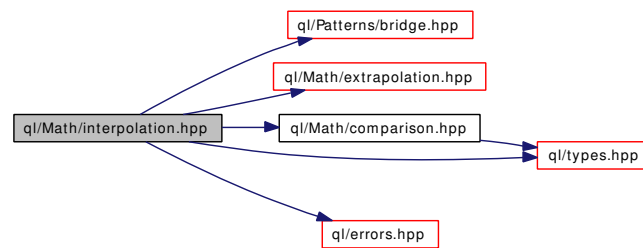
```
#include <ql/Math/extrapolation.hpp>
```

```
#include <ql/Math/comparison.hpp>
```

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

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

Include dependency graph for interpolation.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterpolationImpl](#)  
*abstract base class for interpolation implementations*
- class [Interpolation](#)  
*base class for 1-D interpolations.*
- class [Interpolation::templateImpl](#)  
*basic template implementation*

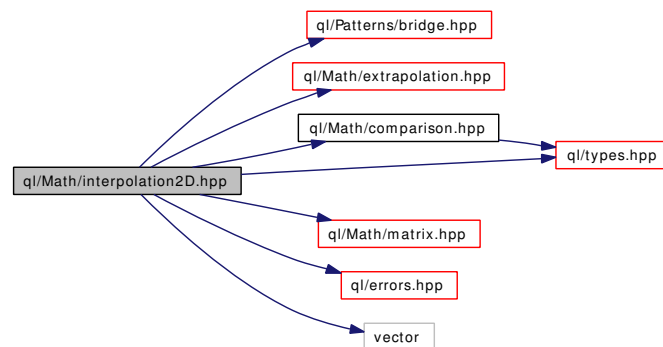
## 8.201 ql/Math/interpolation2D.hpp File Reference

### 8.201.1 Detailed Description

abstract base classes for 2-D interpolations

```
#include <ql/Patterns/bridge.hpp>
#include <ql/Math/extrapolation.hpp>
#include <ql/Math/comparison.hpp>
#include <ql/Math/matrix.hpp>
#include <ql/errors.hpp>
#include <ql/types.hpp>
#include <vector>
```

Include dependency graph for interpolation2D.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Interpolation2DImpl](#)  
*abstract base class for 2-D interpolation implementations*
- class [Interpolation2D](#)  
*base class for 2-D interpolations.*
- class [Interpolation2D::templateImpl](#)  
*basic template implementation*

## 8.202 ql/Math/kronrodintegral.hpp File Reference

### 8.202.1 Detailed Description

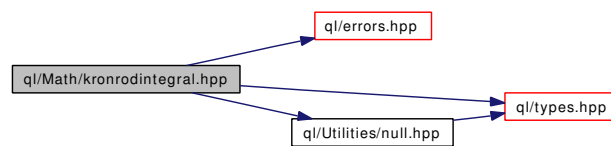
Integral of a 1-dimensional function using the Gauss-Kronrod method.

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

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

```
#include <ql/Utilities/null.hpp>
```

Include dependency graph for kronrodintegral.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [KronrodIntegral](#)  
*Integral of a 1-dimensional function using the Gauss-Kronrod method.*

## 8.203 ql/Math/lexicographicalview.hpp File Reference

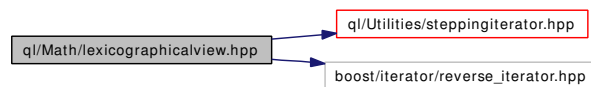
### 8.203.1 Detailed Description

Lexicographical 2-D view of a contiguous set of data.

```
#include <ql/Utilities/steppingiterator.hpp>
```

```
#include <boost/iterator/reverse_iterator.hpp>
```

Include dependency graph for lexicographicalview.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LexicographicalView](#)  
*Lexicographical 2-D view of a contiguous set of data.*

## 8.204 ql/Math/linearinterpolation.hpp File Reference

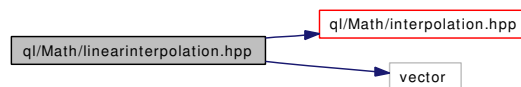
### 8.204.1 Detailed Description

linear interpolation between discrete points

```
#include <ql/Math/interpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for linearinterpolation.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class [LinearInterpolation](#)  
*Linear interpolation between discrete points*
- class [Linear](#)  
*[Linear](#) interpolation factory and traits.*

## 8.205 ql/Math/linearleastsquaresregression.hpp File Reference

### 8.205.1 Detailed Description

general linear least square regression

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

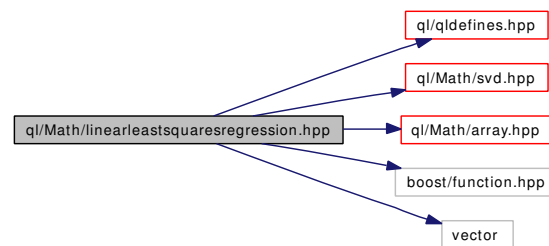
```
#include <ql/Math/svd.hpp>
```

```
#include <ql/Math/array.hpp>
```

```
#include <boost/function.hpp>
```

```
#include <vector>
```

Include dependency graph for linearleastsquaresregression.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LinearLeastSquaresRegression](#)  
*general linear least squares regression*

## 8.206 ql/Math/loglinearinterpolation.hpp File Reference

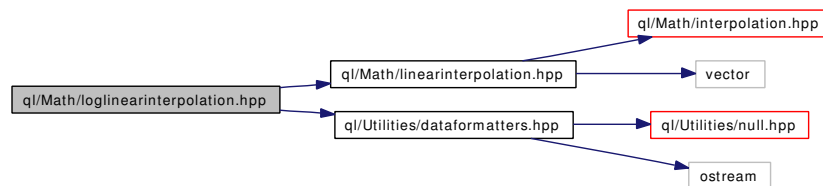
### 8.206.1 Detailed Description

log-linear interpolation between discrete points

```
#include <ql/Math/linearinterpolation.hpp>
```

```
#include <ql/Utilities/dataformatters.hpp>
```

Include dependency graph for loglinearinterpolation.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class [LogLinearInterpolation](#)
- class [LogLinear](#)
  - log-linear interpolation factory and traits*



## 8.207 ql/Math/matrix.hpp File Reference

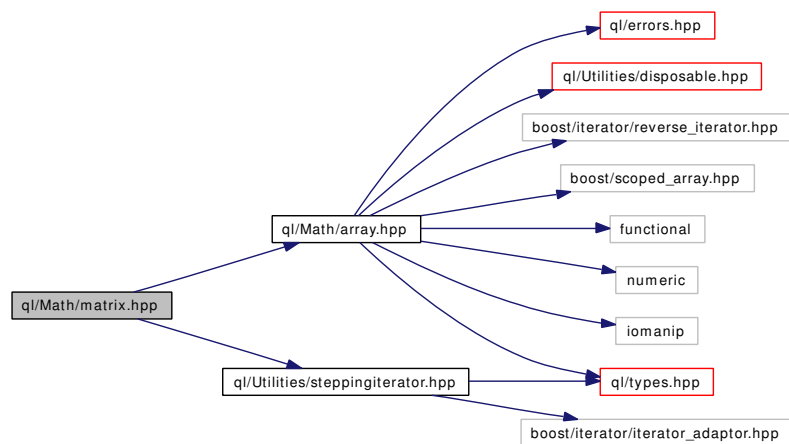
### 8.207.1 Detailed Description

matrix used in linear algebra.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Utilities/steppingiterator.hpp>
```

Include dependency graph for matrix.hpp:



### Namespaces

- namespace `QuantLib`

### Classes

- class `Matrix`

*Matrix used in linear algebra.*

## 8.208 ql/Math/multicubicspline.hpp File Reference

### 8.208.1 Detailed Description

N-dimensional cubic spline interpolation between discrete points.

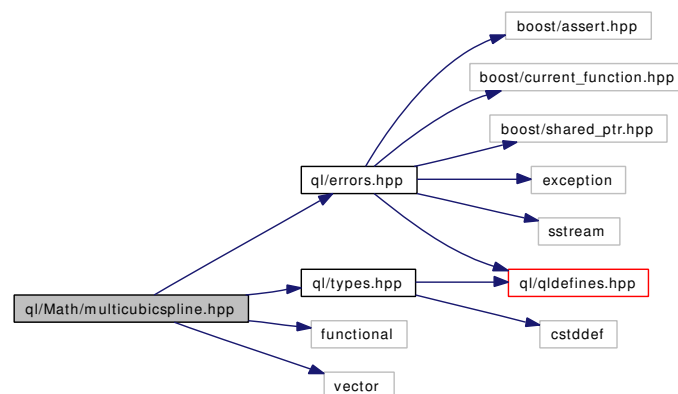
```
#include <ql/errors.hpp>
```

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

```
#include <functional>
```

```
#include <vector>
```

Include dependency graph for multicubicspline.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class [MultiCubicSpline](#)

### Typedefs

- typedef `std::vector< std::vector< Real > >` **SplineGrid**
- typedef `DataTable< Real >` **base\_data\_table**
- typedef `Data< std::vector< Real >, EmptyArg >` **base\_data**
- typedef `Point< Real, EmptyArg >` **base\_arg\_type**
- typedef `Point< Real, EmptyRes >` **base\_return\_type**
- typedef `Point< Size, EmptyDim >` **base\_dimensions**
- typedef `Point< base_data_table, EmptyRes >` **base\_output\_data**
- typedef `base_cubic_spline` **cubic\_spline\_01**
- typedef `n_cubic_spline< cubic_spline_01 >` **cubic\_spline\_02**
- typedef `n_cubic_spline< cubic_spline_02 >` **cubic\_spline\_03**
- typedef `n_cubic_spline< cubic_spline_03 >` **cubic\_spline\_04**

- typedef n\_cubic\_spline< cubic\_spline\_04 > **cubic\_spline\_05**
- typedef n\_cubic\_spline< cubic\_spline\_05 > **cubic\_spline\_06**
- typedef n\_cubic\_spline< cubic\_spline\_06 > **cubic\_spline\_07**
- typedef n\_cubic\_spline< cubic\_spline\_07 > **cubic\_spline\_08**
- typedef n\_cubic\_spline< cubic\_spline\_08 > **cubic\_spline\_09**
- typedef n\_cubic\_spline< cubic\_spline\_09 > **cubic\_spline\_10**
- typedef n\_cubic\_spline< cubic\_spline\_10 > **cubic\_spline\_11**
- typedef n\_cubic\_spline< cubic\_spline\_11 > **cubic\_spline\_12**
- typedef n\_cubic\_spline< cubic\_spline\_12 > **cubic\_spline\_13**
- typedef n\_cubic\_spline< cubic\_spline\_13 > **cubic\_spline\_14**
- typedef n\_cubic\_spline< cubic\_spline\_14 > **cubic\_spline\_15**
- typedef base\_cubic\_splint **cubic\_splint\_01**
- typedef n\_cubic\_splint< cubic\_splint\_01 > **cubic\_splint\_02**
- typedef n\_cubic\_splint< cubic\_splint\_02 > **cubic\_splint\_03**
- typedef n\_cubic\_splint< cubic\_splint\_03 > **cubic\_splint\_04**
- typedef n\_cubic\_splint< cubic\_splint\_04 > **cubic\_splint\_05**
- typedef n\_cubic\_splint< cubic\_splint\_05 > **cubic\_splint\_06**
- typedef n\_cubic\_splint< cubic\_splint\_06 > **cubic\_splint\_07**
- typedef n\_cubic\_splint< cubic\_splint\_07 > **cubic\_splint\_08**
- typedef n\_cubic\_splint< cubic\_splint\_08 > **cubic\_splint\_09**
- typedef n\_cubic\_splint< cubic\_splint\_09 > **cubic\_splint\_10**
- typedef n\_cubic\_splint< cubic\_splint\_10 > **cubic\_splint\_11**
- typedef n\_cubic\_splint< cubic\_splint\_11 > **cubic\_splint\_12**
- typedef n\_cubic\_splint< cubic\_splint\_12 > **cubic\_splint\_13**
- typedef n\_cubic\_splint< cubic\_splint\_13 > **cubic\_splint\_14**
- typedef n\_cubic\_splint< cubic\_splint\_14 > **cubic\_splint\_15**
- typedef detail::SplineGrid **SplineGrid**

## 8.209 ql/Math/normaldistribution.hpp File Reference

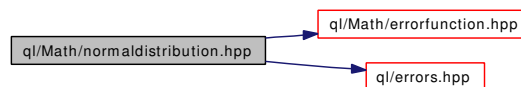
### 8.209.1 Detailed Description

normal, cumulative and inverse cumulative distributions

```
#include <ql/Math/errorfunction.hpp>
```

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

Include dependency graph for normaldistribution.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- 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.*

### Typedefs

- typedef `NormalDistribution` **GaussianDistribution**
- typedef `InverseCumulativeNormal` **InvCumulativeNormalDistribution**

## 8.210 ql/Math/poissondistribution.hpp File Reference

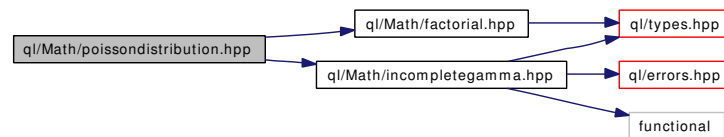
### 8.210.1 Detailed Description

Poisson distribution.

```
#include <ql/Math/factorial.hpp>
```

```
#include <ql/Math/incompletegamma.hpp>
```

Include dependency graph for poissondistribution.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **PoissonDistribution**  
*Normal distribution function.*
- class **CumulativePoissonDistribution**  
*Cumulative Poisson distribution function.*
- class **InverseCumulativePoisson**  
*Inverse cumulative Poisson distribution function.*

## 8.211 ql/Math/primenumbers.hpp File Reference

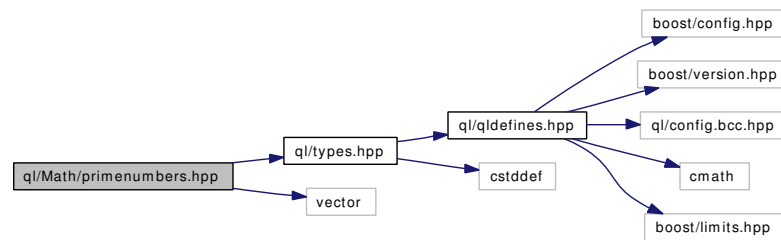
### 8.211.1 Detailed Description

Prime numbers calculator.

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

```
#include <vector>
```

Include dependency graph for primenumbers.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [PrimeNumbers](#)  
*Prime numbers calculator.*

## 8.212 ql/Math/pseudosqrt.hpp File Reference

### 8.212.1 Detailed Description

pseudo square root of a real symmetric matrix

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for pseudosqrt.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct [SalvagingAlgorithm](#)  
*algorithm used for matricial pseudo square root*

## 8.213 ql/Math/riskstatistics.hpp File Reference

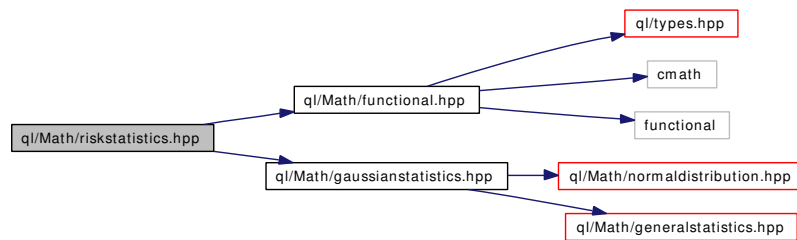
### 8.213.1 Detailed Description

empirical-distribution risk measures

```
#include <ql/Math/functional.hpp>
```

```
#include <ql/Math/gaussianstatistics.hpp>
```

Include dependency graph for riskstatistics.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GenericRiskStatistics](#)  
*empirical-distribution risk measures*

### Typedefs

- typedef `GenericRiskStatistics< GaussianStatistics >` [RiskStatistics](#)  
*default risk measures tool*



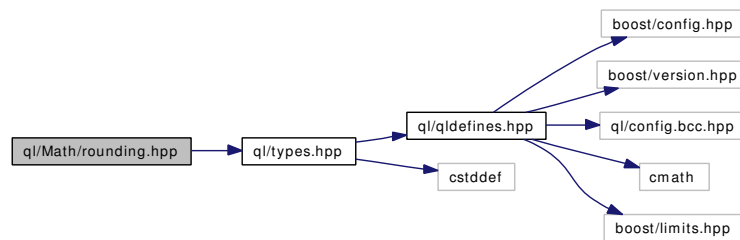
## 8.214 ql/Math/rounding.hpp File Reference

### 8.214.1 Detailed Description

Rounding implementation.

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

Include dependency graph for rounding.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Rounding**  
*basic rounding class*
- class **UpRounding**  
*Up-rounding.*
- class **DownRounding**  
*Down-rounding.*
- class **ClosestRounding**  
*Closest rounding.*
- class **CeilingTruncation**  
*Ceiling truncation.*
- class **FloorTruncation**  
*Floor truncation.*

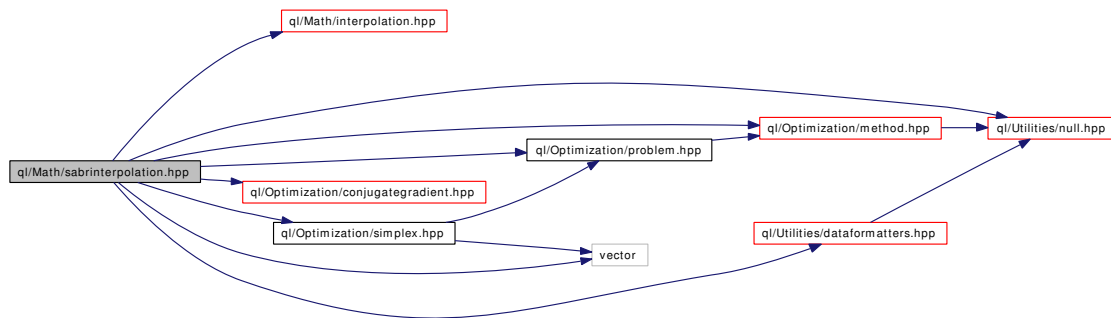
## 8.215 ql/Math/sabrinterpolation.hpp File Reference

### 8.215.1 Detailed Description

SABR interpolation interpolation between discrete points.

```
#include <ql/Math/interpolation.hpp>
#include <ql/Optimization/method.hpp>
#include <ql/Optimization/problem.hpp>
#include <ql/Optimization/conjugategradient.hpp>
#include <ql/Optimization/simplex.hpp>
#include <ql/Utilities/null.hpp>
#include <ql/Utilities/dataformatters.hpp>
#include <vector>
```

Include dependency graph for sabrinterpolation.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class [SABRInterpolation](#)  
*SABR smile interpolation between discrete volatility points.*

### Functions

- Real **unsafeSabrVolatility** (Rate strike, Rate forward, Time expiryTime, Real alpha, Real beta, Real nu, Real rho)
- Real **sabrVolatility** (Rate strike, Rate forward, Time expiryTime, Real alpha, Real beta, Real nu, Real rho)

## 8.216 ql/Math/sampledcurve.hpp File Reference

### 8.216.1 Detailed Description

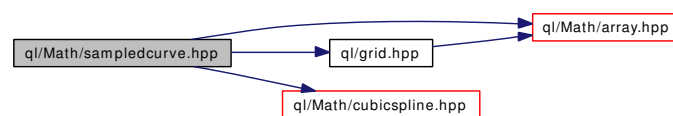
a class that contains a sampled curve

```
#include <ql/Math/array.hpp>
```

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

```
#include <ql/Math/cubicspline.hpp>
```

Include dependency graph for sampledcurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SampledCurve](#)  
*This class contains a sampled curve.*

### Typedefs

- typedef `SampledCurve` **SampledCurveSet**

### Functions

- void **swap** (`SampledCurve &`, `SampledCurve &`)
- `std::ostream &` **operator**<< (`std::ostream &out`, `const SampledCurve &a`)

## 8.217 ql/Math/segmentintegral.hpp File Reference

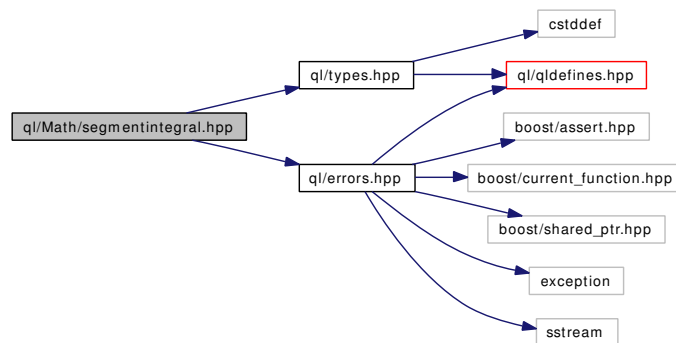
### 8.217.1 Detailed Description

Integral of a one-dimensional function.

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

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

Include dependency graph for segmentintegral.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SegmentIntegral](#)  
*Integral of a one-dimensional function.*

## 8.218 ql/Math/sequencestatistics.hpp File Reference

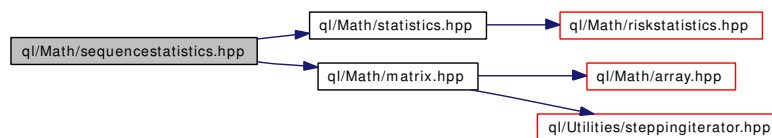
### 8.218.1 Detailed Description

Statistics tools for sequence (vector, list, array) samples.

```
#include <ql/Math/statistics.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for sequencestatistics.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GenericSequenceStatistics](#)  
*Statistics analysis of N-dimensional (sequence) data.*

### Defines

- #define **DEFINE\_SEQUENCE\_STAT\_CONST\_METHOD\_VOID**(METHOD)
- #define **DEFINE\_SEQUENCE\_STAT\_CONST\_METHOD\_DOUBLE**(METHOD)

### Typedefs

- typedef [GenericSequenceStatistics](#) [SequenceStatistics](#)  
*default multi-dimensional statistics tool*

### 8.218.2 Define Documentation

#### 8.218.2.1 #define DEFINE\_SEQUENCE\_STAT\_CONST\_METHOD\_VOID(METHOD)

Value:

```
template <class Stat> \
    std::vector<Real> \
    GenericSequenceStatistics<Stat>::METHOD() const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(); \
        return results_; \
    }
```

### 8.218.2.2 #define DEFINE\_SEQUENCE\_STAT\_CONST\_METHOD\_DOUBLE(METHOD)

**Value:**

```
template <class Stat> \
    std::vector<Real> \
    GenericSequenceStatistics<Stat>::METHOD(Real x) const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(x); \
        return results_; \
    }
```

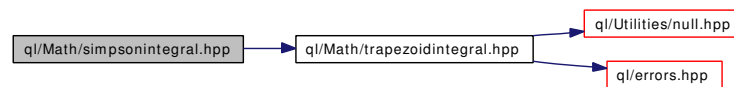
## 8.219 ql/Math/simpsonintegral.hpp File Reference

### 8.219.1 Detailed Description

integral of a one-dimensional function

```
#include <ql/Math/trapezoidintegral.hpp>
```

Include dependency graph for `simpsonintegral.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SimpsonIntegral](#)  
*Integral of a one-dimensional function.*

## 8.220 ql/Math/statistics.hpp File Reference

### 8.220.1 Detailed Description

statistics tool with risk measures

```
#include <ql/Math/riskstatistics.hpp>
```

Include dependency graph for statistics.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef RiskStatistics [Statistics](#)  
*default statistics tool*



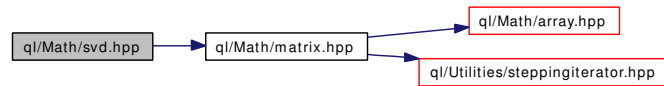
## 8.221 ql/Math/svd.hpp File Reference

### 8.221.1 Detailed Description

singular value decomposition

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for svd.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **SVD**  
*Singular value decomposition.*

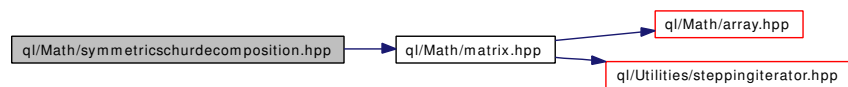
## 8.222 `ql/Math/symmetricschurdecomposition.hpp` File Reference

### 8.222.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for `symmetricschurdecomposition.hpp`:



### Namespaces

- namespace `QuantLib`

### Classes

- class [SymmetricSchurDecomposition](#)  
*symmetric threshold Jacobi algorithm.*

## 8.223 ql/Math/tqreigendecomposition.hpp File Reference

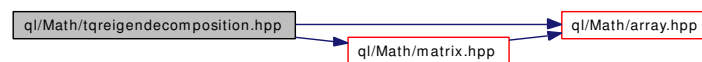
### 8.223.1 Detailed Description

tridiag. QR eigen decomposition with explicite shift aka Wilkinson

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for tqreigendecomposition.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TqrEigenDecomposition](#)  
*tridiag. QR eigen decomposition with explicite shift aka Wilkinson*

## 8.224 ql/Math/transformedgrid.hpp File Reference

### 8.224.1 Detailed Description

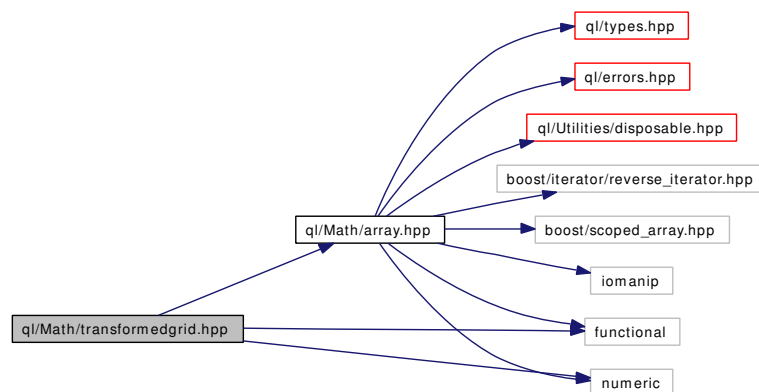
encapsulates a grid

```
#include <ql/Math/array.hpp>
```

```
#include <functional>
```

```
#include <numeric>
```

Include dependency graph for transformedgrid.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TransformedGrid](#)  
*transformed grid*

## 8.225 ql/Math/trapezoidintegral.hpp File Reference

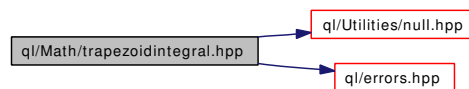
### 8.225.1 Detailed Description

integral of a one-dimensional function

```
#include <ql/Utilities/null.hpp>
```

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

Include dependency graph for trapezoidintegral.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TrapezoidIntegral](#)  
*Integral of a one-dimensional function.*

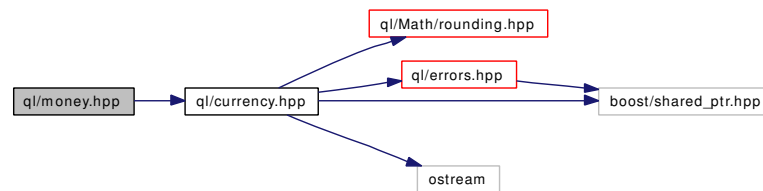
## 8.226 ql/money.hpp File Reference

### 8.226.1 Detailed Description

cash amount in a given currency

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

Include dependency graph for money.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Money](#)  
*amount of cash*

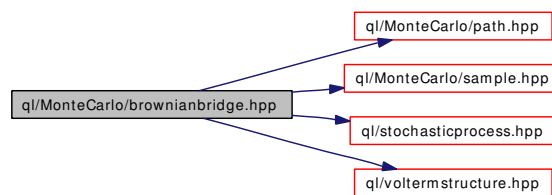
## 8.227 ql/MonteCarlo/brownianbridge.hpp File Reference

### 8.227.1 Detailed Description

Browian bridge.

```
#include <ql/MonteCarlo/path.hpp>
#include <ql/MonteCarlo/sample.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/voltermstructure.hpp>
```

Include dependency graph for brownianbridge.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BrownianBridge**  
*Builds Wiener process paths using Gaussian variates.*

## 8.228 ql/MonteCarlo/earlyexercisepathpricer.hpp File Reference

### 8.228.1 Detailed Description

base class for early exercise single-path pricers

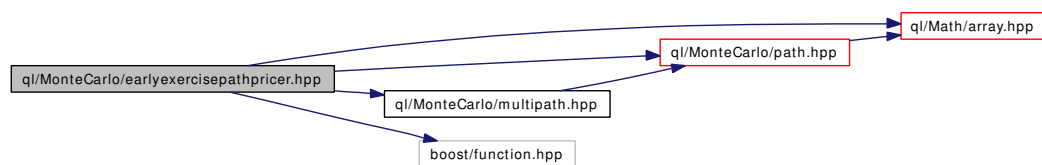
```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/path.hpp>
```

```
#include <ql/MonteCarlo/multipath.hpp>
```

```
#include <boost/function.hpp>
```

Include dependency graph for earlyexercisepathpricer.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **EarlyExercisePathPricer**  
*base class for early exercise path pricers*



## 8.229 ql/MonteCarlo/getcovariance.hpp File Reference

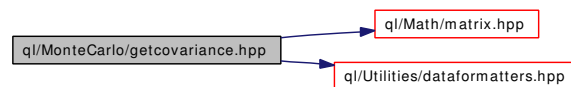
### 8.229.1 Detailed Description

Covariance matrix calculation.

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Utilities/dataformatters.hpp>
```

Include dependency graph for getcovariance.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CovarianceDecomposition](#)

### Functions

- `template<class DataIterator> Disposable< Matrix > getCovariance (DataIterator volBegin, DataIterator volEnd, const Matrix &corr, Real tolerance=1.0e-12)`

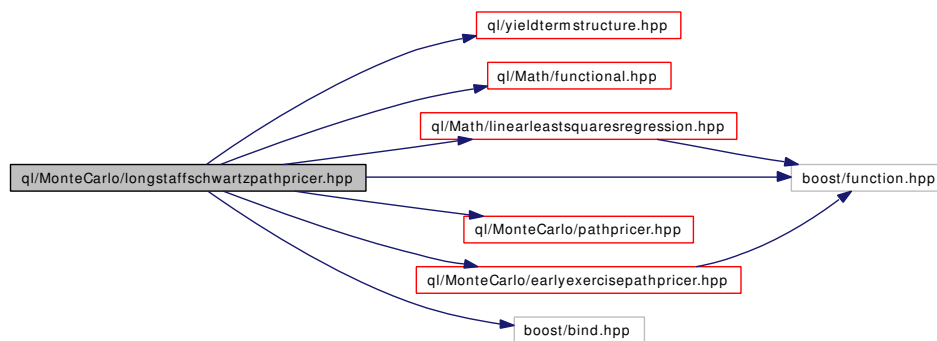
## 8.230 ql/MonteCarlo/longstaffschwartzpathpricer.hpp File Reference

### 8.230.1 Detailed Description

Longstaff-Schwarz path pricer for early exercise options.

```
#include <ql/yieldtermstructure.hpp>
#include <ql/Math/functional.hpp>
#include <ql/Math/linearleastsquaresregression.hpp>
#include <ql/MonteCarlo/pathpricer.hpp>
#include <ql/MonteCarlo/earlyexercisepathpricer.hpp>
#include <boost/bind.hpp>
#include <boost/function.hpp>
```

Include dependency graph for longstaffschwartzpathpricer.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LongstaffSchwartzPathPricer](#)  
*Longstaff-Schwarz path pricer for early exercise options.*

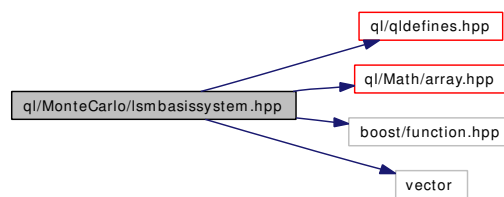
## 8.231 ql/MonteCarlo/lsmbasissystem.hpp File Reference

### 8.231.1 Detailed Description

utility classes for longstaff schwartz early exercise Monte Carlo

```
#include <ql/qldefines.hpp>
#include <ql/Math/array.hpp>
#include <boost/function.hpp>
#include <vector>
```

Include dependency graph for lsmbasissystem.hpp:



### Namespaces

- namespace **QuantLib**

## 8.232 ql/MonteCarlo/mctraits.hpp File Reference

### 8.232.1 Detailed Description

Monte Carlo policies.

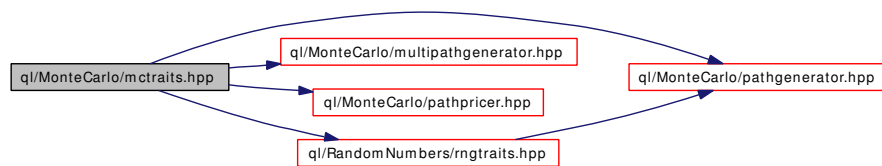
```
#include <ql/MonteCarlo/pathgenerator.hpp>
```

```
#include <ql/MonteCarlo/multipathgenerator.hpp>
```

```
#include <ql/MonteCarlo/pathpricer.hpp>
```

```
#include <ql/RandomNumbers/rngtraits.hpp>
```

Include dependency graph for mctraits.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct [SingleVariate](#)  
*default Monte Carlo traits for single-variate models*
- struct [MultiVariate](#)  
*default Monte Carlo traits for multi-variate models*

## 8.233 ql/MonteCarlo/mctypedefs.hpp File Reference

### 8.233.1 Detailed Description

Default choices for template instantiations.

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

Include dependency graph for mctypedefs.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `MonteCarloModel< SingleVariate< PseudoRandom > >` [OneFactorMonteCarloOption](#)  
*default choice for one-factor Monte Carlo model.*
- typedef `MonteCarloModel< MultiVariate< PseudoRandom > >` [MultiFactorMonteCarloOption](#)  
*default choice for multi-factor Monte Carlo model.*

## 8.234 ql/MonteCarlo/montecarlomodel.hpp File Reference

### 8.234.1 Detailed Description

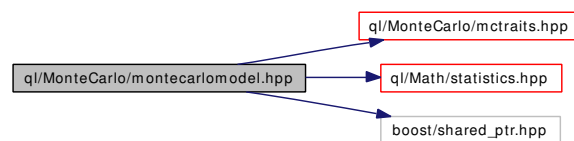
General purpose Monte Carlo model.

```
#include <ql/MonteCarlo/mctraits.hpp>
```

```
#include <ql/Math/statistics.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for montecarlomodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MonteCarloModel](#)  
*General purpose Monte Carlo model for path samples.*

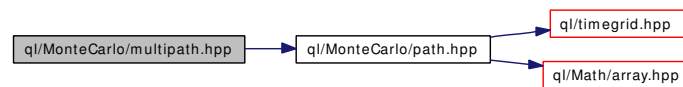
## 8.235 ql/MonteCarlo/multipath.hpp File Reference

### 8.235.1 Detailed Description

Correlated multiple asset paths.

```
#include <ql/MonteCarlo/path.hpp>
```

Include dependency graph for multipath.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **MultiPath**  
*Correlated multiple asset paths.*

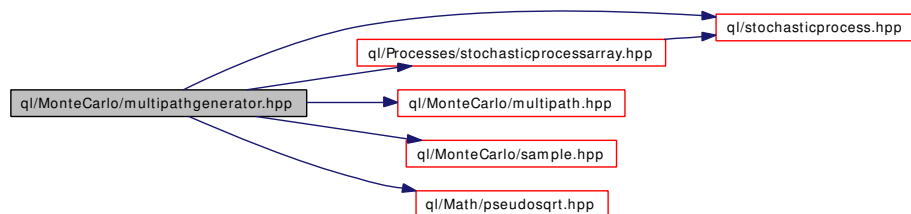
## 8.236 ql/MonteCarlo/multipathgenerator.hpp File Reference

### 8.236.1 Detailed Description

Generates a multi path from a random-array generator.

```
#include <ql/stochasticprocess.hpp>
#include <ql/Processes/stochasticprocessarray.hpp>
#include <ql/MonteCarlo/multipath.hpp>
#include <ql/MonteCarlo/sample.hpp>
#include <ql/Math/pseudosqrt.hpp>
```

Include dependency graph for multipathgenerator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **MultiPathGenerator**  
*Generates a multipath from a random number generator.*



## 8.237 ql/MonteCarlo/path.hpp File Reference

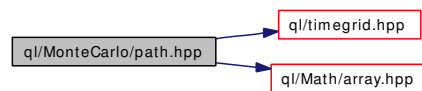
### 8.237.1 Detailed Description

single factor random walk

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

```
#include <ql/Math/array.hpp>
```

Include dependency graph for path.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Path](#)

## 8.238 ql/MonteCarlo/pathgenerator.hpp File Reference

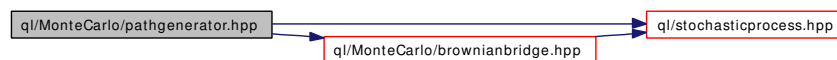
### 8.238.1 Detailed Description

Generates random paths using a sequence generator.

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

```
#include <ql/MonteCarlo/brownianbridge.hpp>
```

Include dependency graph for pathgenerator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [PathGenerator](#)  
*Generates random paths using a sequence generator.*

## 8.239 ql/MonteCarlo/pathpricer.hpp File Reference

### 8.239.1 Detailed Description

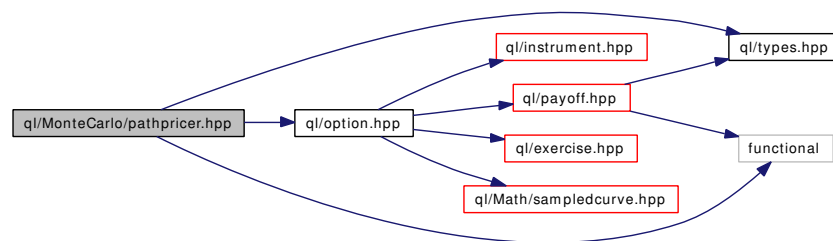
base class for single-path pricers

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

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

```
#include <functional>
```

Include dependency graph for pathpricer.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **PathPricer**  
*base class for path pricers*

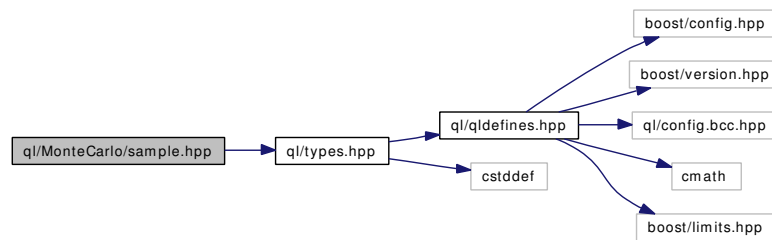
## 8.240 ql/MonteCarlo/sample.hpp File Reference

### 8.240.1 Detailed Description

weighted sample

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

Include dependency graph for sample.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct [Sample](#)  
*weighted sample*

## 8.241 ql/numericalmethod.hpp File Reference

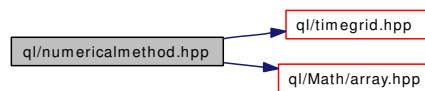
### 8.241.1 Detailed Description

Numerical method class.

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

```
#include <ql/Math/array.hpp>
```

Include dependency graph for numericalmethod.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [NumericalMethod](#)  
*Numerical method (tree, finite-differences) base class.*

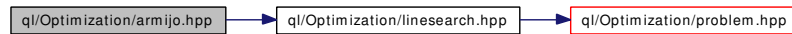
## 8.242 ql/Optimization/armijo.hpp File Reference

### 8.242.1 Detailed Description

Armijo line-search class.

```
#include <ql/Optimization/linesearch.hpp>
```

Include dependency graph for armijo.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ArmijoLineSearch](#)  
*Armijo line search.*

## 8.243 ql/Optimization/conjugategradient.hpp File Reference

### 8.243.1 Detailed Description

Conjugate gradient optimization method.

```
#include <ql/Optimization/armijo.hpp>
```

Include dependency graph for conjugategradient.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ConjugateGradient](#)  
*Multi-dimensional Conjugate Gradient class.*

## 8.244 ql/Optimization/constraint.hpp File Reference

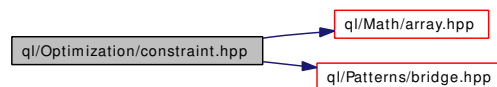
### 8.244.1 Detailed Description

Abstract constraint class.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Patterns/bridge.hpp>
```

Include dependency graph for constraint.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ConstraintImpl](#)  
*Base class for constraint implementations.*
- class [Constraint](#)  
*Base constraint class.*
- class [NoConstraint](#)  
*No constraint.*
- class [PositiveConstraint](#)  
*Constraint imposing positivity to all arguments*
- class [BoundaryConstraint](#)  
*Constraint imposing all arguments to be in [low,high]*
- class [CompositeConstraint](#)  
*Constraint enforcing both given sub-constraints*



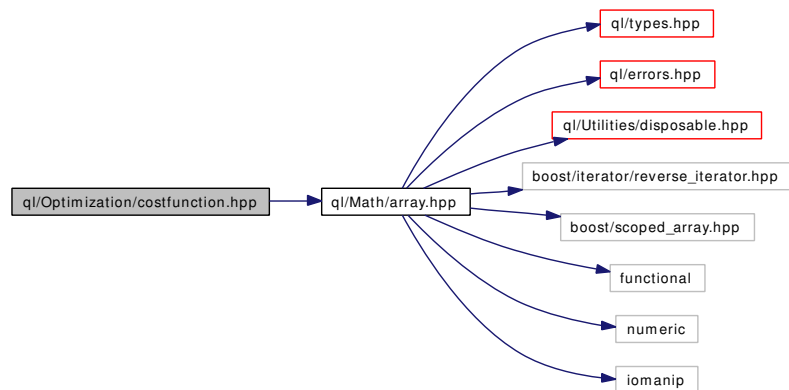
## 8.245 ql/Optimization/costfunction.hpp File Reference

### 8.245.1 Detailed Description

Optimization cost function class.

```
#include <ql/Math/array.hpp>
```

Include dependency graph for costfunction.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CostFunction](#)  
*Cost function abstract class for optimization problem.*

## 8.246 ql/Optimization/criteria.hpp File Reference

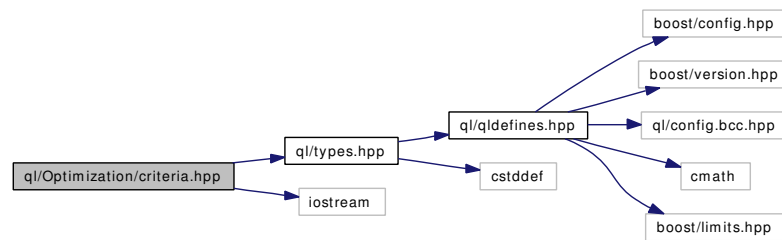
### 8.246.1 Detailed Description

Optimization criteria class.

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

```
#include <iostream>
```

Include dependency graph for criteria.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [EndCriteria](#)  
*Criteria to end optimization process.*

### Functions

- `std::ostream & operator<< (std::ostream &out, EndCriteria::Type ec)`

## 8.247 ql/Optimization/leastsquare.hpp File Reference

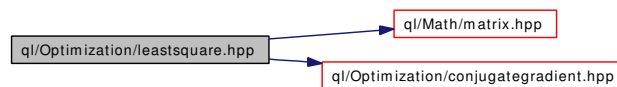
### 8.247.1 Detailed Description

Least square cost function.

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Optimization/conjugategradient.hpp>
```

Include dependency graph for leastsquare.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LeastSquareProblem](#)  
*Base class for least square problem.*
- class [LeastSquareFunction](#)  
*Cost function for least-square problems.*
- class [NonLinearLeastSquare](#)  
*Non-linear least-square method.*

## 8.248 ql/Optimization/levenbergmarquardt.hpp File Reference

### 8.248.1 Detailed Description

Levenberg-Marquardt optimization method.

```
#include <ql/Optimization/problem.hpp>
```

Include dependency graph for levenbergmarquardt.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LevenbergMarquardt](#)  
*Levenberg-Marquardt optimization method.*

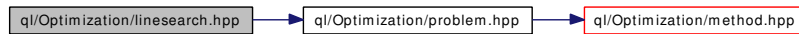
## 8.249 ql/Optimization/linesearch.hpp File Reference

### 8.249.1 Detailed Description

Line search abstract class.

```
#include <ql/Optimization/problem.hpp>
```

Include dependency graph for linesearch.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LineSearch](#)  
*Base class for line search.*

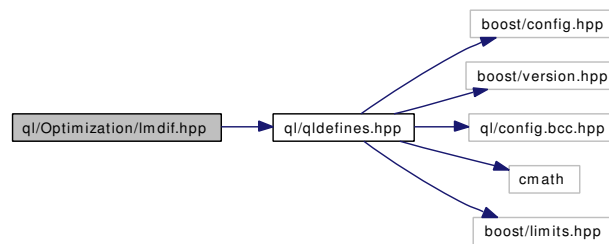
## 8.250 ql/Optimization/lmdif.hpp File Reference

### 8.250.1 Detailed Description

wrapper for MINPACK minimization routine

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

Include dependency graph for lmdif.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::MINPACK**

### Functions

- void **lmdif** (int m, int n, double \*x, double \*fvec, double ftol, double xtol, double gtol, int maxfev, double epsfcn, double \*diag, int mode, double factor, int nprint, int \*info, int \*nfev, double \*fjac, int ldfjac, int \*ipvt, double \*qtf, double \*wa1, double \*wa2, double \*wa3, double \*wa4)

## 8.251 ql/Optimization/method.hpp File Reference

### 8.251.1 Detailed Description

Abstract optimization method class.

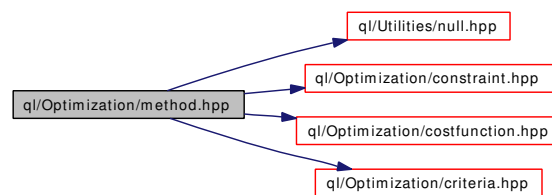
```
#include <ql/Utilities/null.hpp>
```

```
#include <ql/Optimization/constraint.hpp>
```

```
#include <ql/Optimization/costfunction.hpp>
```

```
#include <ql/Optimization/criteria.hpp>
```

Include dependency graph for method.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [OptimizationMethod](#)  
*Abstract class for constrained optimization method.*

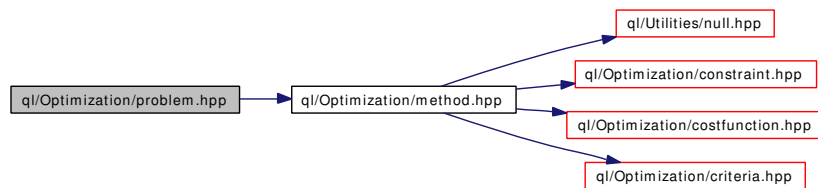
## 8.252 ql/Optimization/problem.hpp File Reference

### 8.252.1 Detailed Description

Abstract optimization class.

```
#include <ql/Optimization/method.hpp>
```

Include dependency graph for problem.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Problem**  
*Constrained optimization problem.*



## 8.253 ql/Optimization/simplex.hpp File Reference

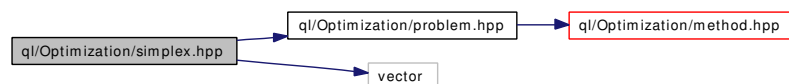
### 8.253.1 Detailed Description

Simplex optimization method.

```
#include <ql/Optimization/problem.hpp>
```

```
#include <vector>
```

Include dependency graph for simplex.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Simplex](#)  
*Multi-dimensional simplex class.*

## 8.254 ql/Optimization/steepestdescent.hpp File Reference

### 8.254.1 Detailed Description

Steepest descent optimization method.

```
#include <ql/Optimization/armijo.hpp>
```

Include dependency graph for steepestdescent.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SteepestDescent](#)  
*Multi-dimensional steepest-descent class.*

## 8.255 ql/option.hpp File Reference

### 8.255.1 Detailed Description

Base option class.

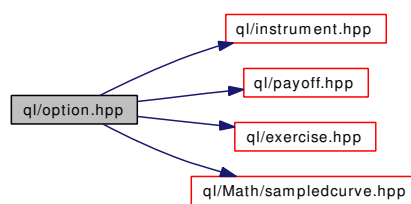
```
#include <ql/instrument.hpp>
```

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

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

```
#include <ql/Math/sampledcurve.hpp>
```

Include dependency graph for option.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Option**  
*base option class*
- class **Option::arguments**
- class **PriceCurve**  
*additional pricing results*
- class **Greeks**  
*additional option results*
- class **MoreGreeks**  
*more additional option results*

## 8.256 ql/Patterns/bridge.hpp File Reference

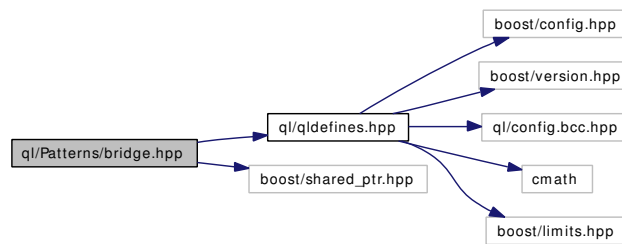
### 8.256.1 Detailed Description

bridge pattern (a.k.a. handle-body idiom)

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

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for bridge.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Bridge](#)  
*The Bridge pattern made explicit.*

## 8.257 ql/Patterns/composite.hpp File Reference

### 8.257.1 Detailed Description

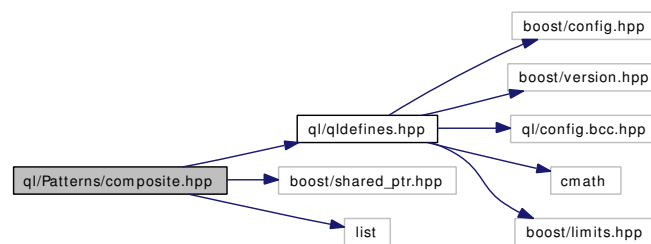
composite pattern

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

```
#include <boost/shared_ptr.hpp>
```

```
#include <list>
```

Include dependency graph for composite.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Composite](#)  
*Composite pattern.*

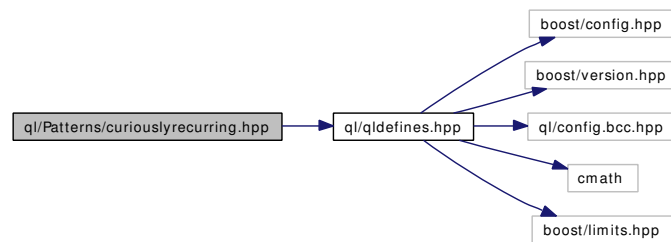
## 8.258 ql/Patterns/curiouslyrecurring.hpp File Reference

### 8.258.1 Detailed Description

Curiously recurring template pattern.

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

Include dependency graph for curiouslyrecurring.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CuriouslyRecurringTemplate](#)  
*Support for the curiously recurring template pattern.*

## 8.259 ql/Patterns/lazyobject.hpp File Reference

### 8.259.1 Detailed Description

framework for calculation on demand and result caching

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for lazyobject.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LazyObject](#)

*Framework for calculation on demand and result caching.*

## 8.260 ql/Patterns/observable.hpp File Reference

### 8.260.1 Detailed Description

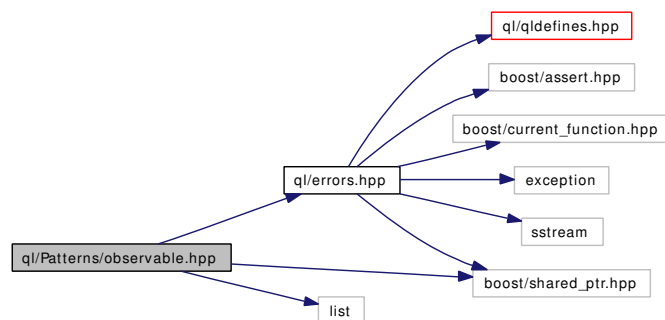
observer/observable pattern

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

```
#include <boost/shared_ptr.hpp>
```

```
#include <list>
```

Include dependency graph for observable.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Observable](#)  
*Object that notifies its changes to a set of observables.*
- class [Observer](#)  
*Object that gets notified when a given observable changes.*



## 8.261 ql/Patterns/singleton.hpp File Reference

### 8.261.1 Detailed Description

basic support for the singleton pattern

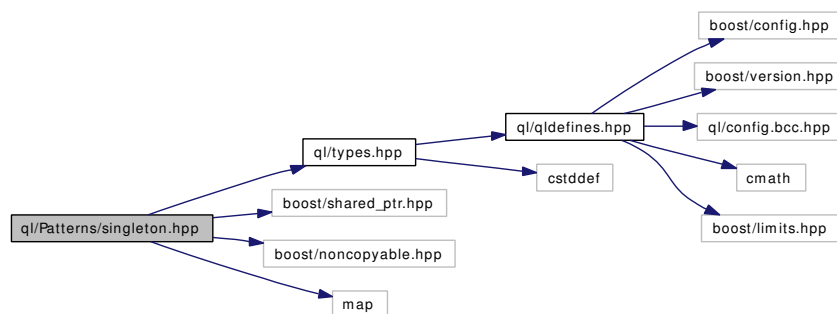
```
#include <ql/types.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

```
#include <boost/noncopyable.hpp>
```

```
#include <map>
```

Include dependency graph for singleton.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [Singleton](#)

*Basic support for the singleton pattern.*

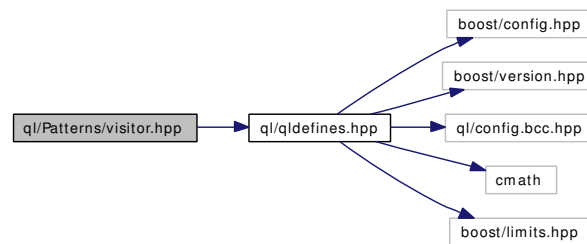
## 8.262 ql/Patterns/visitor.hpp File Reference

### 8.262.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

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

Include dependency graph for visitor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AcyclicVisitor](#)  
*degenerate base class for the Acyclic Visitor pattern*
- class [Visitor](#)  
*Visitor for a specific class*

## 8.263 ql/payoff.hpp File Reference

### 8.263.1 Detailed Description

Option payoff classes.

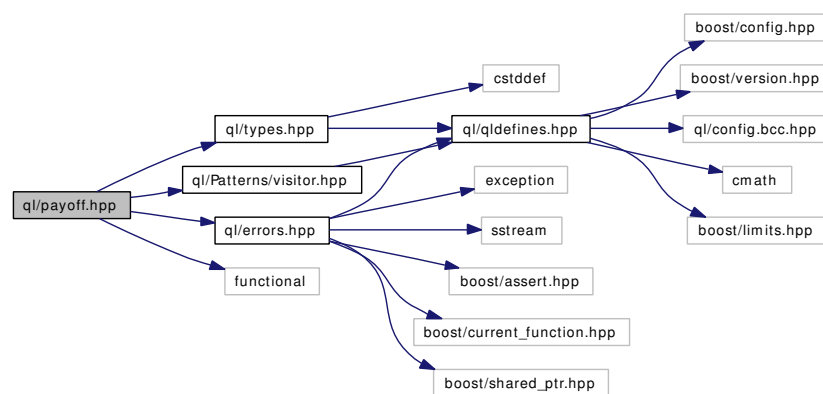
```
#include <ql/types.hpp>
```

```
#include <ql/Patterns/visitor.hpp>
```

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

```
#include <functional>
```

Include dependency graph for payoff.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Payoff](#)  
*Base class for option payoffs.*

## 8.264 ql/period.hpp File Reference

### 8.264.1 Detailed Description

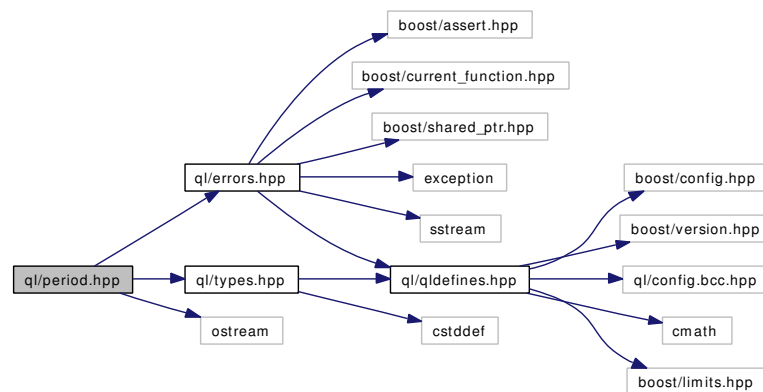
period- and frequency-related classes and enumerations

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

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

```
#include <ostream>
```

Include dependency graph for period.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

### Classes

- class [Period](#)  
*Time period described by a number of a given time unit.*

### Enumerations

- enum [Frequency](#) {  
[NoFrequency](#) = -1, [Once](#) = 0, [Annual](#) = 1, [Semiannual](#) = 2,  
[EveryFourthMonth](#) = 3, [Quarterly](#) = 4, [Bimonthly](#) = 6, [Monthly](#) = 12,  
[Biweekly](#) = 26, [Weekly](#) = 52, [Daily](#) = 365 }  
*Frequency of events.*
- enum [TimeUnit](#) { **Days**, **Weeks**, **Months**, **Years** }  
*Units used to describe time periods.*

## Functions

- `std::ostream & operator<<` (`std::ostream &`, `const long_period_holder &`)
- `std::ostream & operator<<` (`std::ostream &`, `const short_period_holder &`)
- `detail::long_period_holder` [long\\_period](#) (`const Period &`)  
*output periods in long format (e.g. "2 weeks")*
- `detail::short_period_holder` [short\\_period](#) (`const Period &`)  
*output periods in short format (e.g. "2w")*
- `Period operator *` (`Integer n`, `TimeUnit units`)
- `Period operator *` (`TimeUnit units`, `Integer n`)
- `Period operator-` (`const Period &p`)
- `Period operator *` (`Integer n`, `const Period &p`)
- `Period operator *` (`const Period &p`, `Integer n`)
- `bool operator==` (`const Period &p1`, `const Period &p2`)
- `bool operator!=` (`const Period &p1`, `const Period &p2`)
- `bool operator>` (`const Period &p1`, `const Period &p2`)
- `bool operator<=` (`const Period &p1`, `const Period &p2`)
- `bool operator>=` (`const Period &p1`, `const Period &p2`)

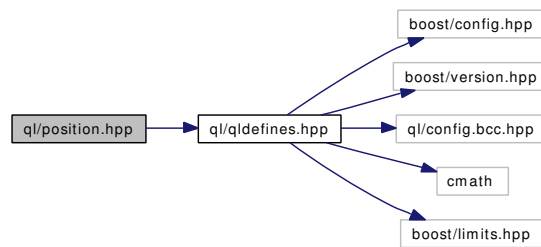
## 8.265 ql/position.hpp File Reference

### 8.265.1 Detailed Description

Short or long position.

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

Include dependency graph for position.hpp:



### Namespaces

- namespace **QuantLib**

## 8.266 ql/Pricers/discretegeometricaso.hpp File Reference

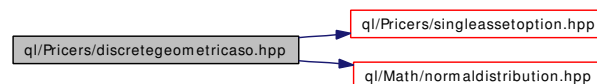
### 8.266.1 Detailed Description

Discrete Geometric Average Strike Option.

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for discretegeometricaso.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DiscreteGeometricASO](#)  
*Discrete geometric average-strike Asian option (European style).*

## 8.267 ql/Pricers/mccliquestoption.hpp File Reference

### 8.267.1 Detailed Description

Cliquet option priced with Monte Carlo simulation.

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

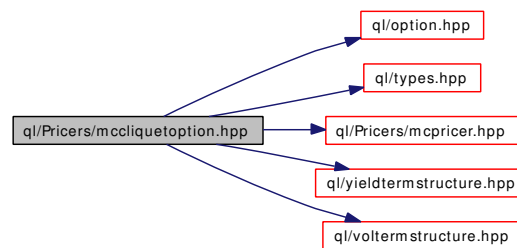
```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

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

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

Include dependency graph for mccliquestoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **McCliquetOption**  
*simple example of Monte Carlo pricer*



## 8.268 ql/Pricers/mcdiscretearithmeticsaso.hpp File Reference

### 8.268.1 Detailed Description

Discrete Arithmetic Average Strike Option.

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

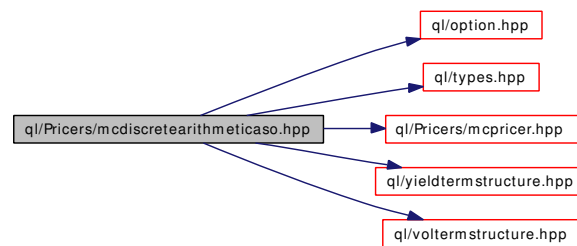
```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

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

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

Include dependency graph for mcdiscretearithmeticsaso.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [McDiscreteArithmeticASO](#)  
*example of Monte Carlo pricer using a control variate.*

## 8.269 ql/Pricers/mceverest.hpp File Reference

### 8.269.1 Detailed Description

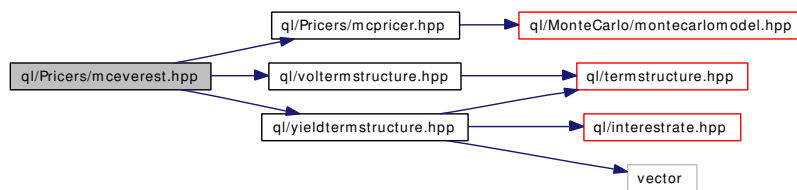
Everest-type option pricer

```
#include <ql/Pricers/mcpricer.hpp>
```

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

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

Include dependency graph for mceverest.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **McEverest**  
*Everest-type option pricer.*

## 8.270 ql/Pricers/mchimalaya.hpp File Reference

### 8.270.1 Detailed Description

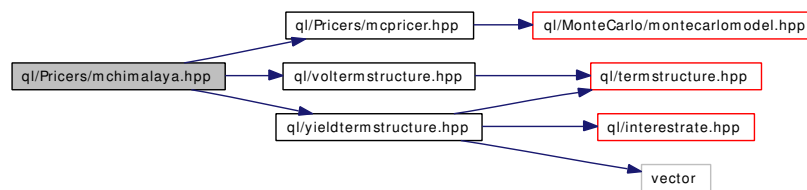
Himalayan-type option pricer.

```
#include <ql/Pricers/mcpricer.hpp>
```

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

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

Include dependency graph for mchimalaya.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **McHimalaya**  
*Himalayan-type option pricer.*

## 8.271 ql/Pricers/mcmaxbasket.hpp File Reference

### 8.271.1 Detailed Description

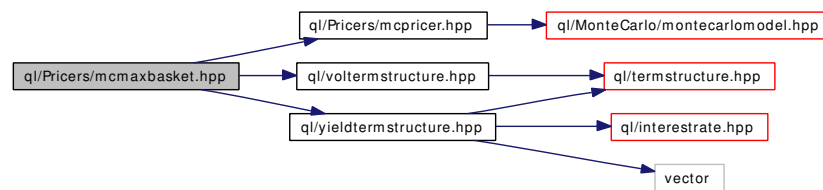
Max Basket Monte Carlo pricer.

```
#include <ql/Pricers/mcpricer.hpp>
```

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

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

Include dependency graph for mcmaxbasket.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class `McMaxBasket`  
*simple example of multi-factor Monte Carlo pricer*

## 8.272 ql/Pricers/mcpagoda.hpp File Reference

### 8.272.1 Detailed Description

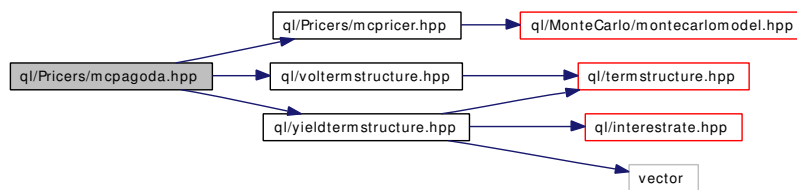
Roofed multi asset Asian option.

```
#include <ql/Pricers/mcpricer.hpp>
```

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

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

Include dependency graph for mcpagoda.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **McPagoda**  
*roofed Asian option*

## 8.273 ql/Pricers/mcperformanceoption.hpp File Reference

### 8.273.1 Detailed Description

Performance option priced with Monte Carlo simulation.

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

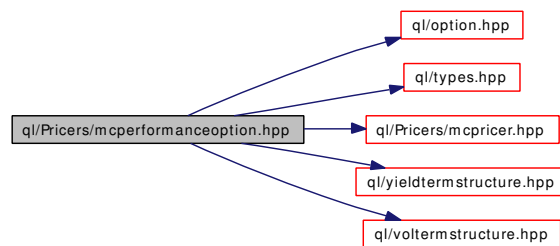
```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

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

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

Include dependency graph for mcperformanceoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [McPerformanceOption](#)  
*Performance option computed using Monte Carlo simulation.*

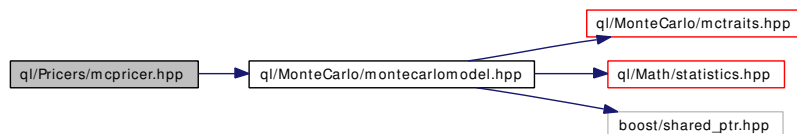
## 8.274 ql/Pricers/mcpricer.hpp File Reference

### 8.274.1 Detailed Description

base class for Monte Carlo pricers

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

Include dependency graph for mcpricer.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [McPricer](#)  
*base class for Monte Carlo pricers*

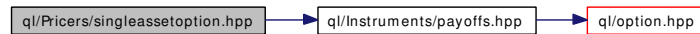
## 8.275 ql/Pricers/singleassetoption.hpp File Reference

### 8.275.1 Detailed Description

common code for option evaluation

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for singleassetoption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SingleAssetOption](#)  
*Black-Scholes-Merton option.*



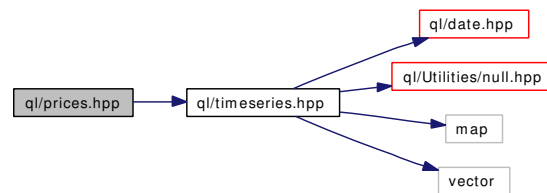
## 8.276 ql/prices.hpp File Reference

### 8.276.1 Detailed Description

price classes

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

Include dependency graph for prices.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [IntervalPrice](#)  
*interval price*

### Enumerations

- enum **PriceType** {  
    [Bid](#), [Ask](#), [Last](#), [Close](#),  
    [Mid](#), [MidEquivalent](#), [MidRobust](#) }  
*Price types.*

### Functions

- Real [midEquivalent](#) (const Real bid, const Real ask, const Real last, const Real close)
- Real [midRobust](#) (const Real bid, const Real ask)

## 8.277 ql/pricingengine.hpp File Reference

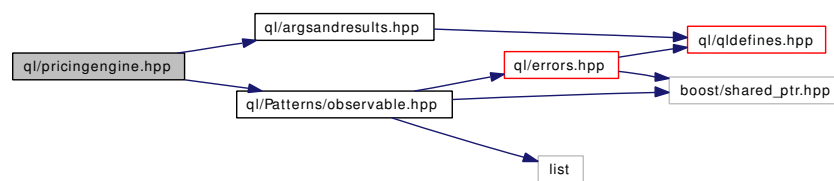
### 8.277.1 Detailed Description

Base class for pricing engines.

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

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for pricingengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [PricingEngine](#)  
*interface for pricing engines*
- class [GenericEngine](#)  
*template base class for option pricing engines*

## 8.278 ql/PricingEngines/americanpayoffatexpiry.hpp File Reference

### 8.278.1 Detailed Description

Analytical formulae for american exercise with payoff at expiry.

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for americanpayoffatexpiry.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AmericanPayoffAtExpiry](#)

## 8.279 ql/PricingEngines/americanpayoffathit.hpp File Reference

### 8.279.1 Detailed Description

Analytical formulae for american exercise with payoff at hit.

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for americanpayoffathit.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AmericanPayoffAtHit](#)

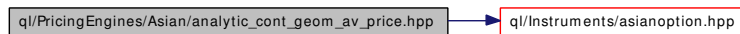
## 8.280 ql/PricingEngines/Asian/analytic\_cont\_geom\_av\_price.hpp File Reference

### 8.280.1 Detailed Description

Analytic engine for continuous geometric average price Asian.

```
#include <ql/Instruments/asianoption.hpp>
```

Include dependency graph for analytic\_cont\_geom\_av\_price.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticContinuousGeometricAveragePriceAsianEngine](#)  
*Pricing engine for European continuous geometric average price Asian.*

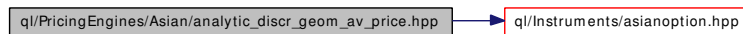
## 8.281 ql/PricingEngines/Asian/analytic\_discr\_geom\_av\_price.hpp File Reference

### 8.281.1 Detailed Description

Analytic engine for discrete geometric average price Asian.

```
#include <ql/Instruments/asianoption.hpp>
```

Include dependency graph for analytic\_discr\_geom\_av\_price.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticDiscreteGeometricAveragePriceAsianEngine](#)  
*Pricing engine for European discrete geometric average price Asian.*

## 8.282 ql/PricingEngines/Asian/mc\_discr\_arith\_av\_price.hpp File Reference

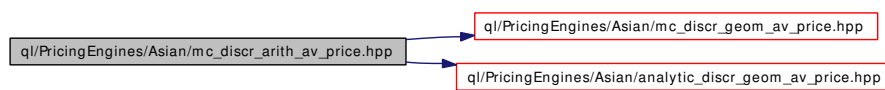
### 8.282.1 Detailed Description

Monte Carlo engine for discrete arithmetic average price Asian.

```
#include <ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp>
```

```
#include <ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp>
```

Include dependency graph for mc\_discr\_arith\_av\_price.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCDiscreteArithmeticAPEngine](#)

*Monte Carlo pricing engine for discrete arithmetic average price Asian.*

## 8.283 ql/PricingEngines/Asian/mc\_discr\_geom\_av\_price.hpp File Reference

### 8.283.1 Detailed Description

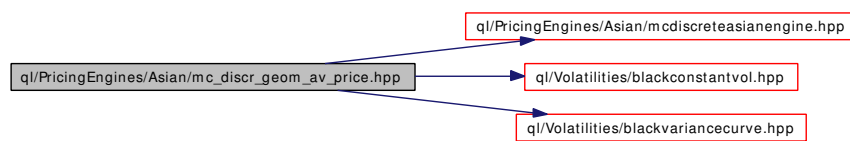
Monte Carlo engine for discrete geometric average price Asian.

```
#include <ql/PricingEngines/Asian/mcdiscreteasianengine.hpp>
```

```
#include <ql/Volatilities/blackconstantvol.hpp>
```

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for mc\_discr\_geom\_av\_price.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [MCDiscreteGeometricAPEngine](#)

*Monte Carlo pricing engine for discrete geometric average price Asian.*



## 8.284 ql/PricingEngines/Asian/mcdiscreteasianengine.hpp File Reference

### 8.284.1 Detailed Description

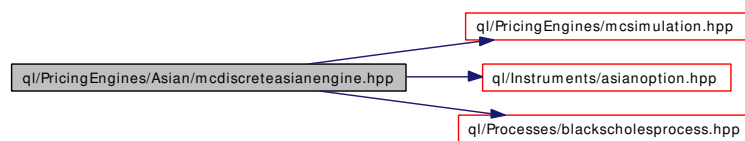
Monte Carlo pricing engine for discrete average Asians.

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Instruments/asianoption.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcdiscreteasianengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCDiscreteAveragingAsianEngine](#)

*Pricing engine for discrete average Asians using Monte Carlo simulation.*

## 8.285 ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference

### 8.285.1 Detailed Description

Analytic barrier option engines.

```
#include <ql/Instruments/barrieroption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for analyticbarrierengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticBarrierEngine](#)  
*Pricing engine for barrier options using analytical formulae.*

## 8.286 ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference

### 8.286.1 Detailed Description

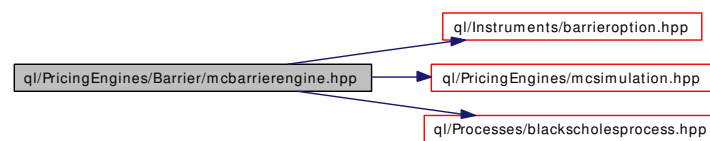
Monte Carlo barrier option engines.

```
#include <ql/Instruments/barrieroption.hpp>
```

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcbarrierengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCBarrierEngine](#)

*Pricing engine for barrier options using Monte Carlo simulation.*

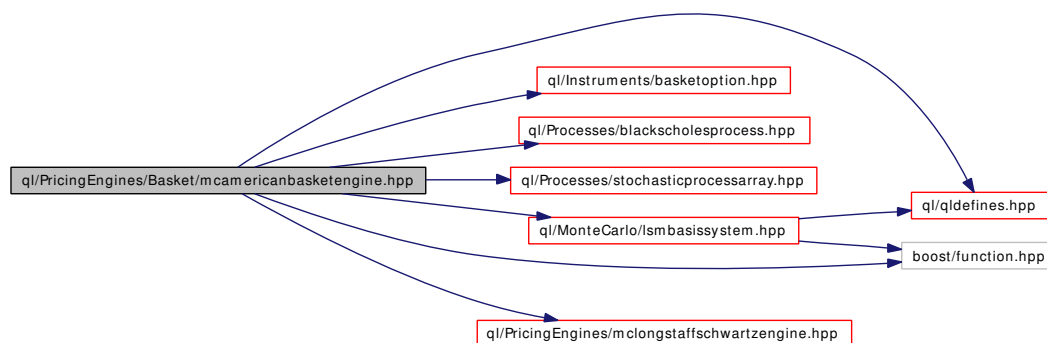
## 8.287 ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference

### 8.287.1 Detailed Description

Least-square Monte Carlo engines.

```
#include <ql/qldefines.hpp>
#include <ql/Instruments/basketoption.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
#include <ql/Processes/stochasticprocessarray.hpp>
#include <ql/MonteCarlo/lsmbasissystem.hpp>
#include <ql/PricingEngines/mclongstaffschwartzengine.hpp>
#include <boost/function.hpp>
```

Include dependency graph for mcamericanbasketengine.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **MCAmericanBasketEngine**  
*least-square Monte Carlo engine*

## 8.288 ql/PricingEngines/Basket/mcbasketengine.hpp File Reference

### 8.288.1 Detailed Description

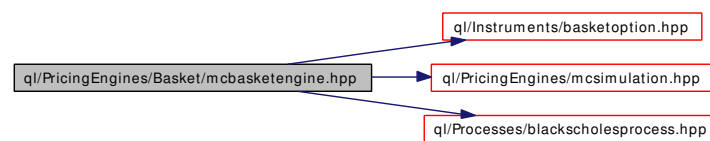
European basket MC Engine.

```
#include <ql/Instruments/basketoption.hpp>
```

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcbasketengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **MCBasketEngine**

*Pricing engine for basket options using Monte Carlo simulation.*

## 8.289 ql/PricingEngines/Basket/stulzengine.hpp File Reference

### 8.289.1 Detailed Description

2D European Basket formulae, due to Stulz (1982)

```
#include <ql/Instruments/basketoption.hpp>
```

Include dependency graph for stulzengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [StulzEngine](#)  
*Pricing engine for 2D European Baskets.*

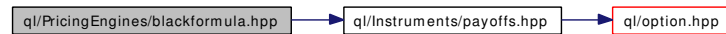
## 8.290 ql/PricingEngines/blackformula.hpp File Reference

### 8.290.1 Detailed Description

Black formula.

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for blackformula.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackFormula**  
*Black-formula calculator.*

## 8.291 ql/PricingEngines/blackmodel.hpp File Reference

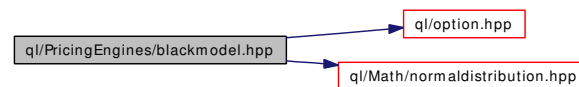
### 8.291.1 Detailed Description

Black formula and associated functions.

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

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for blackmodel.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Functions

- Real [blackFormula](#) (Option::Type optionType, Real strike, Real forward, Real stdDev)
- Real [blackImpliedStdDevApproximation](#) (Option::Type optionType, Real strike, Real forward, Real blackPrice)
- Real [blackImpliedStdDev](#) (Option::Type optionType, Real strike, Real forward, Real blackPrice, Real guess=Null< Real >(), Real accuracy=1.0e-6)
- Real **blackItmProbability** (Option::Type optionType, Real strike, Real forward, Real stdDev)
- Real [blackFormula](#) (Real forward, Real strike, Real stdDev, Option::Type optionType)
- Real [itmBlackProbability](#) (Real forward, Real strike, Real stdDev, Option::Type optionType)



## 8.292 ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference

### 8.292.1 Detailed Description

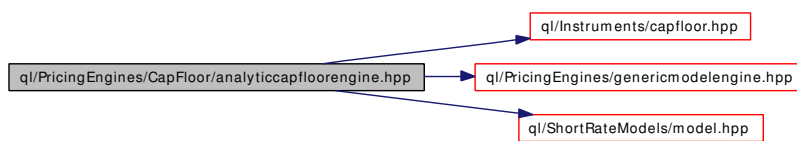
Analytic engine for caps/floors.

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

```
#include <ql/ShortRateModels/model.hpp>
```

Include dependency graph for analyticcapfloorengine.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [AnalyticCapFloorEngine](#)  
*Analytic engine for cap/floor.*

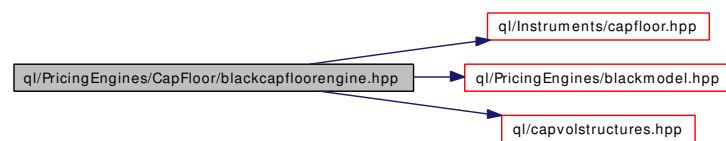
## 8.293 ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference

### 8.293.1 Detailed Description

Black-formula cap/floor engine.

```
#include <ql/Instruments/capfloor.hpp>
#include <ql/PricingEngines/blackmodel.hpp>
#include <ql/capvolstructures.hpp>
```

Include dependency graph for blackcapfloorengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackCapFloorEngine**  
*Black-formula cap/floor engine.*

## 8.294 ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference

### 8.294.1 Detailed Description

discretized cap/floor

```
#include <ql/Instruments/capfloor.hpp>
```

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

Include dependency graph for discretizedcapfloor.hpp:



### Namespaces

- namespace **QuantLib**

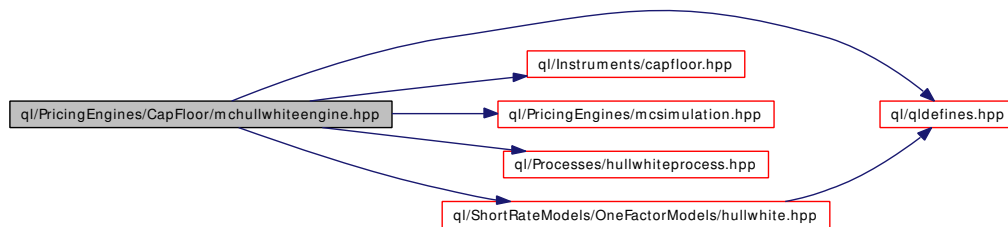
## 8.295 ql/PricingEngines/CapFloor/mchullwhiteengine.hpp File Reference

### 8.295.1 Detailed Description

Monte Carlo Hull-White engine for cap/floors.

```
#include <ql/qldefines.hpp>
#include <ql/Instruments/capfloor.hpp>
#include <ql/PricingEngines/mcsimulation.hpp>
#include <ql/Processes/hullwhiteprocess.hpp>
#include <ql/ShortRateModels/OneFactorModels/hullwhite.hpp>
```

Include dependency graph for mchullwhiteengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCHullWhiteCapFloorEngine](#)  
*Monte Carlo Hull-White engine for cap/floors.*
- class [MakeMCHullWhiteCapFloorEngine](#)  
*Monte Carlo Hull-White cap-floor engine factory.*

## 8.296 ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference

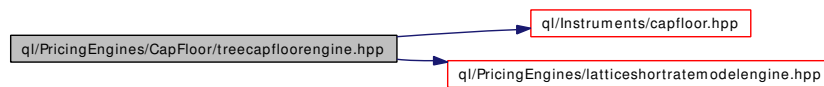
### 8.296.1 Detailed Description

Numerical lattice engine for cap/floors.

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

Include dependency graph for treecapfloorengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TreeCapFloorEngine](#)  
*Numerical lattice engine for cap/floors.*

## 8.297 ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference

### 8.297.1 Detailed Description

Analytic Cliquet engine.

```
#include <ql/Instruments/cliquetoption.hpp>
```

Include dependency graph for analyticcliquetengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticCliquetEngine](#)  
*Pricing engine for Cliquet options using analytical formulae.*

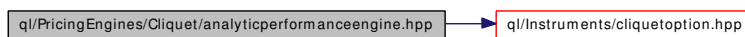
## 8.298 ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference

### 8.298.1 Detailed Description

Analytic performance engine.

```
#include <ql/Instruments/cliquetoption.hpp>
```

Include dependency graph for analyticperformanceengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticPerformanceEngine](#)  
*Pricing engine for performance options using analytical formulae.*

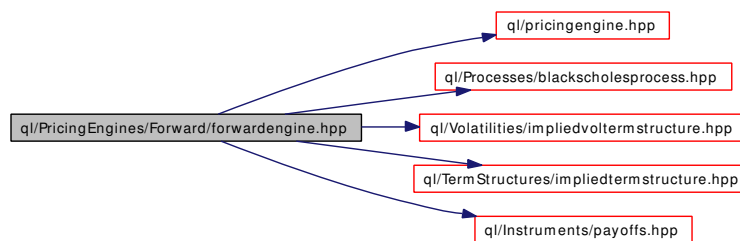
## 8.299 ql/PricingEngines/Forward/forwardengine.hpp File Reference

### 8.299.1 Detailed Description

Forward (strike-resetting) option engine.

```
#include <ql/pricingengine.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
#include <ql/Volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp>
#include <ql/TermStructures/IMPLIEDTERMSTRUCTURE.hpp>
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for forwardengine.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [ForwardOptionArguments](#)  
*Arguments for forward (strike-resetting) option calculation*
- class [ForwardEngine](#)  
*Forward engine base class.*



## 8.300 ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference

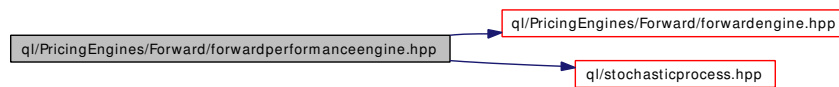
### 8.300.1 Detailed Description

Forward (strike-resetting) performance option engines.

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

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

Include dependency graph for forwardperformanceengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ForwardPerformanceEngine](#)  
*Forward performance engine.*

## 8.301 ql/PricingEngines/Forward/mcvarianceswapengine.hpp File Reference

### 8.301.1 Detailed Description

Monte Carlo variance-swap engine.

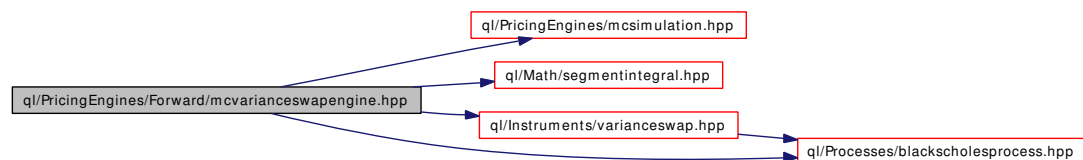
```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Math/segmentintegral.hpp>
```

```
#include <ql/Instruments/varianceswap.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcvarianceswapengine.hpp:



## Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

## Classes

- class **MCVarianceSwapEngine**  
*Variance-swap pricing engine using Monte Carlo simulation,.*
- class **MakeMCVarianceSwapEngine**  
*Monte Carlo variance-swap engine factory.*

## 8.302 ql/PricingEngines/Forward/replicatingvarianceswapengine.hpp File Reference

### 8.302.1 Detailed Description

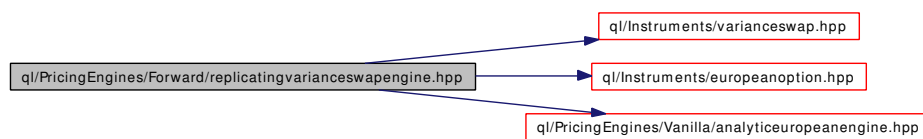
Replicating engine for variance swaps.

```
#include <ql/Instruments/varianceswap.hpp>
```

```
#include <ql/Instruments/europeanoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp>
```

Include dependency graph for replicatingvarianceswapengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ReplicatingVarianceSwapEngine](#)  
*Variance-swap pricing engine using replicating cost,.*

## 8.303 ql/PricingEngines/genericmodelengine.hpp File Reference

### 8.303.1 Detailed Description

Generic option engine based on a model.

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

Include dependency graph for genericmodelengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GenericModelEngine](#)  
*Base class for some pricing engine on a particular model.*

## 8.304 ql/PricingEngines/greeks.hpp File Reference

### 8.304.1 Detailed Description

default greek calculations

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for greeks.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- Real [blackScholesTheta](#) (const boost::shared\_ptr< GeneralizedBlackScholesProcess > &, Real value, Real delta, Real gamma)  
*default theta calculation for Black-Scholes options*
- Real [defaultThetaPerDay](#) (Real theta)  
*default theta-per-day calculation*

## 8.305 ql/PricingEngines/Hybrid/binomialconvertibleengine.hpp File Reference

### 8.305.1 Detailed Description

binomial engine for convertible bonds

```
#include <ql/Lattices/tflattice.hpp>
```

```
#include <ql/PricingEngines/Hybrid/discretizedconvertible.hpp>
```

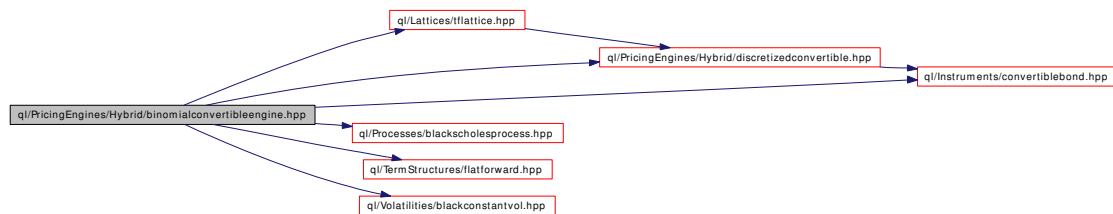
```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/TermStructures/flatforward.hpp>
```

```
#include <ql/Volatilities/blackconstantvol.hpp>
```

```
#include <ql/Instruments/convertiblebond.hpp>
```

Include dependency graph for binomialconvertibleengine.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **BinomialConvertibleEngine**

*Binomial Tsiveriotis-Fernandes engine for convertible bonds.*

## 8.306 ql/PricingEngines/Hybrid/discretizedconvertible.hpp File Reference

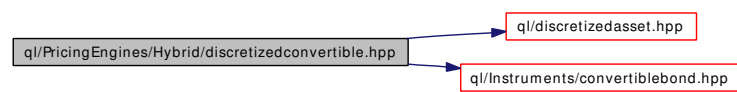
### 8.306.1 Detailed Description

discretized convertible

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

```
#include <ql/Instruments/convertiblebond.hpp>
```

Include dependency graph for discretizedconvertible.hpp:



### Namespaces

- namespace **QuantLib**

## 8.307 ql/PricingEngines/latticeshortratemodelengine.hpp File Reference

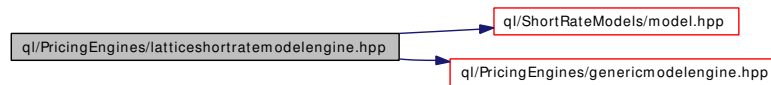
### 8.307.1 Detailed Description

Engine for a short-rate model specialized on a lattice.

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for latticeshortratemodelengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LatticeShortRateModelEngine](#)  
*Engine for a short-rate model specialized on a lattice.*



## 8.308 ql/PricingEngines/Lookback/analyticcontinuousfixedlookback.hpp File Reference

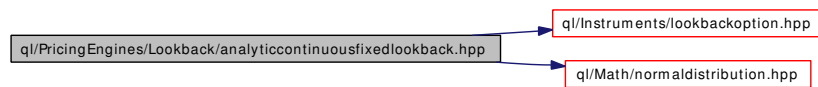
### 8.308.1 Detailed Description

Analytic engine for continuous fixed-strike lookback.

```
#include <ql/Instruments/lookbackoption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for analyticcontinuousfixedlookback.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticContinuousFixedLookbackEngine](#)  
*Pricing engine for European continuous fixed-strike lookback.*

## 8.309 ql/PricingEngines/Lookback/analyticcontinuousfloatinglookback.hpp File Reference

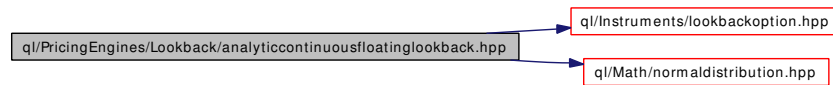
### 8.309.1 Detailed Description

Analytic engine for continuous floating-strike lookback.

```
#include <ql/Instruments/lookbackoption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for analyticcontinuousfloatinglookback.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticContinuousFloatingLookbackEngine](#)  
*Pricing engine for European continuous floating-strike lookback.*

## 8.310 ql/PricingEngines/mclongstaffschwartzengine.hpp File Reference

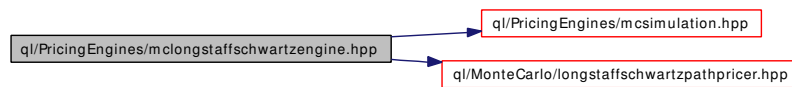
### 8.310.1 Detailed Description

Longstaff Schwartz Monte Carlo engine for early exercise options.

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/MonteCarlo/longstaffschwartzpathpricer.hpp>
```

Include dependency graph for mclongstaffschwartzengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCLongstaffSchwartzEngine](#)  
*Longstaff Schwarz Monte Carlo engine for early exercise options.*

## 8.311 ql/PricingEngines/mcsimulation.hpp File Reference

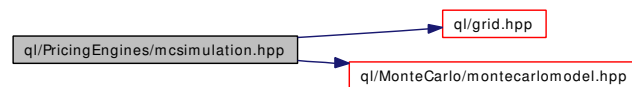
### 8.311.1 Detailed Description

framework for Monte Carlo engines

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

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

Include dependency graph for mcsimulation.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [McSimulation](#)  
*base class for Monte Carlo engines*

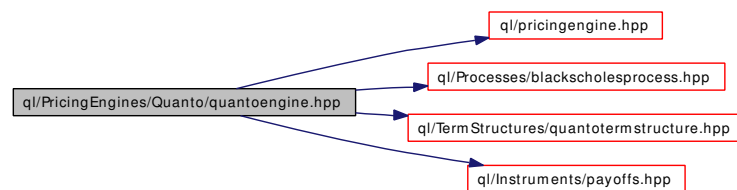
## 8.312 ql/PricingEngines/Quanto/quantoengine.hpp File Reference

### 8.312.1 Detailed Description

Quanto option engine.

```
#include <ql/pricingengine.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
#include <ql/TermStructures/quantotermstructure.hpp>
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for quantoengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [QuantoOptionArguments](#)  
*Arguments for quanto option calculation*
- class [QuantoOptionResults](#)  
*Results from quanto option calculation*
- class [QuantoEngine](#)  
*Quanto engine base class.*

## 8.313 ql/PricingEngines/Swaption/blackswaptionengine.hpp File Reference

### 8.313.1 Detailed Description

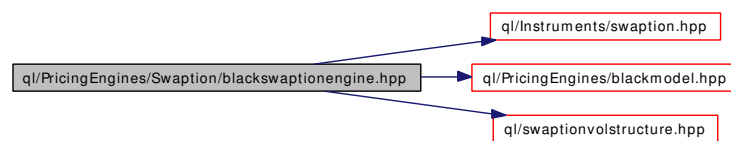
Black-formula swaption engine.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/PricingEngines/blackmodel.hpp>
```

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

Include dependency graph for blackswaptionengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackSwaptionEngine**  
*Black-formula swaption engine.*

## 8.314 ql/PricingEngines/Swaption/discretizedswaption.hpp File Reference

### 8.314.1 Detailed Description

Discretized swaption class.

```
#include <ql/Instruments/swaption.hpp>
```

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

Include dependency graph for discretizedswaption.hpp:



### Namespaces

- namespace **QuantLib**

## 8.315 ql/PricingEngines/Swaption/g2swaptionengine.hpp File Reference

### 8.315.1 Detailed Description

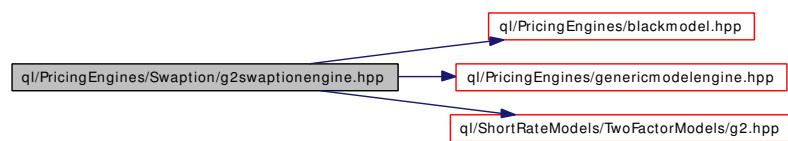
Swaption pricing engine for two-factor additive Gaussian Model G2++.

```
#include <ql/PricingEngines/blackmodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Include dependency graph for g2swaptionengine.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [G2SwaptionEngine](#)  
*Swaption priced by means of the Black formula*



## 8.316 ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp File Reference

### 8.316.1 Detailed Description

Swaption engine using Jamshidian's decomposition.

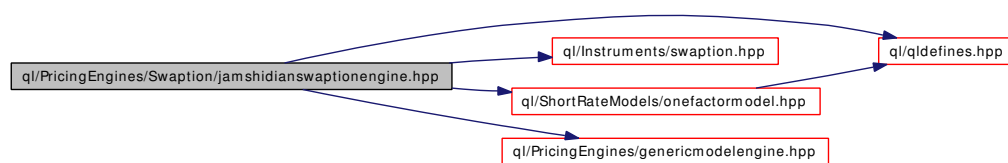
```
#include <ql/qldefines.hpp>
```

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for jamshidianswaptionengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **JamshidianSwaptionEngine**  
*Jamshidian swaption engine.*

## 8.317 ql/PricingEngines/Swaption/lfmswaptionengine.hpp File Reference

### 8.317.1 Detailed Description

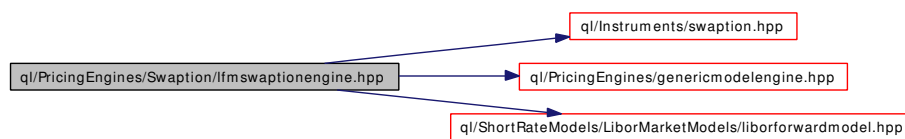
libor forward model swaption engine based on black formula

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

```
#include <ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp>
```

Include dependency graph for lfmswaptionengine.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [LfmSwaptionEngine](#)  
*libor forward model swaption engine based on black formula*

## 8.318 ql/PricingEngines/Swaption/treeswaptionengine.hpp File Reference

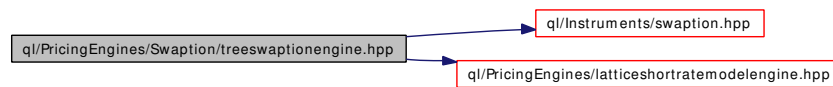
### 8.318.1 Detailed Description

Numerical lattice engines for swaps and swaptions.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

Include dependency graph for treeswaptionengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TreeVanillaSwapEngine](#)  
*Numerical lattice engine for simple swaps.*
- class [TreeSwaptionEngine](#)  
*Numerical lattice engine for swaptions.*

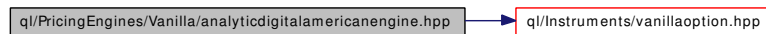
## 8.319 ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference

### 8.319.1 Detailed Description

analytic digital American option engine

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for analyticdigitalamericanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticDigitalAmericanEngine](#)

## 8.320 ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp File Reference

### 8.320.1 Detailed Description

Analytic discrete-dividend European engine.

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

Include dependency graph for analyticdividendeuropeanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticDividendEuropeanEngine](#)  
*Analytic pricing engine for European options with discrete dividends.*

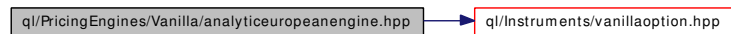
## 8.321 ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference

### 8.321.1 Detailed Description

Analytic European engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for analyticeuropeanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticEuropeanEngine](#)  
*Pricing engine for European vanilla options using analytical formulae.*

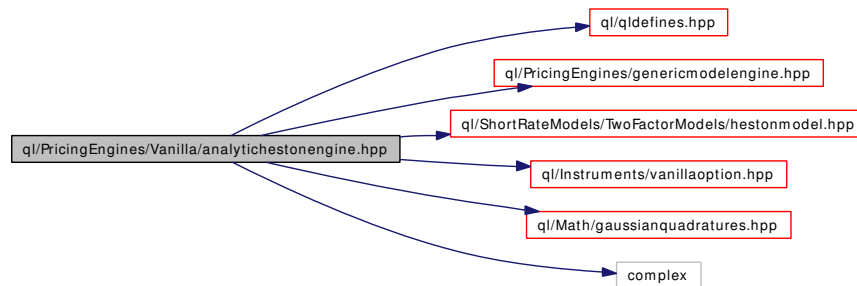
## 8.322 ql/PricingEngines/Vanilla/analytichestonengine.hpp File Reference

### 8.322.1 Detailed Description

analytic Heston-model engine

```
#include <ql/qldefines.hpp>
#include <ql/PricingEngines/genericmodelengine.hpp>
#include <ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp>
#include <ql/Instruments/vanillaoption.hpp>
#include <ql/Math/gaussianquadratures.hpp>
#include <complex>
```

Include dependency graph for analytichestonengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticHestonEngine](#)  
*analytic Heston-model engine based on Fourier transform*

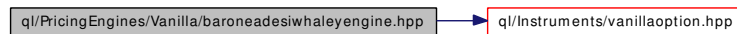
## 8.323 ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference

### 8.323.1 Detailed Description

Barone-Adesi and Whaley approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for baroneadesiwhaleyengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BaroneAdesiWhaleyApproximationEngine](#)



## 8.324 ql/PricingEngines/Vanilla/batesengine.hpp File Reference

### 8.324.1 Detailed Description

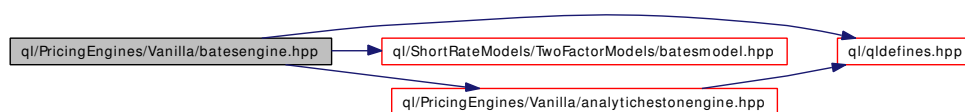
analytic Bates model engine

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

```
#include <ql/ShortRateModels/TwoFactorModels/batesmodel.hpp>
```

```
#include <ql/PricingEngines/Vanilla/analytichestonengine.hpp>
```

Include dependency graph for batesengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BatesEngine](#)  
*Bates model engines based on Fourier transform.*

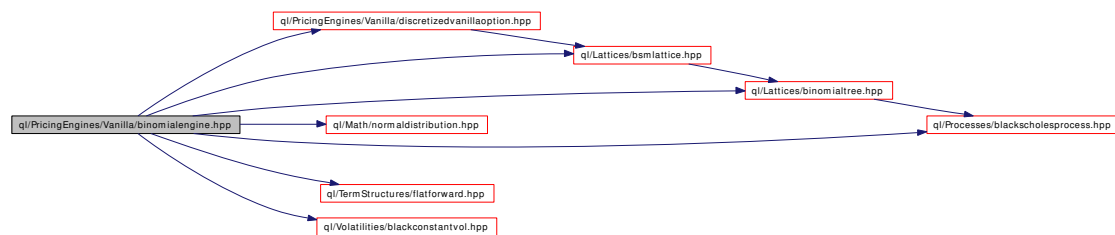
## 8.325 ql/PricingEngines/Vanilla/binomialengine.hpp File Reference

### 8.325.1 Detailed Description

Binomial option engine.

```
#include <ql/Lattices/binomialtree.hpp>
#include <ql/Lattices/bsmlattice.hpp>
#include <ql/Math/normaldistribution.hpp>
#include <ql/PricingEngines/Vanilla/discretizedvanilloption.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
#include <ql/TermStructures/flatforward.hpp>
#include <ql/Volatilities/blackconstantvol.hpp>
```

Include dependency graph for binomialengine.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **BinomialVanillaEngine**  
*Pricing engine for vanilla options using binomial trees.*

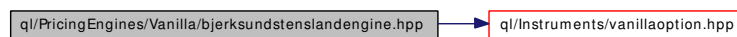
## 8.326 ql/PricingEngines/Vanilla/bjersundstenslandengine.hpp File Reference

### 8.326.1 Detailed Description

Bjersund and Stensland approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for bjersundstenslandengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BjersundStenslandApproximationEngine](#)

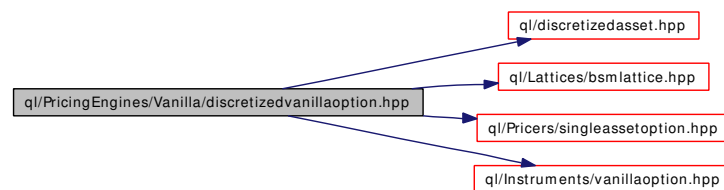
## 8.327 ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference

### 8.327.1 Detailed Description

discretized vanilla option

```
#include <ql/discretizedasset.hpp>
#include <ql/Lattices/bsmlattice.hpp>
#include <ql/Pricers/singleassetoption.hpp>
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for discretizedvanillaoption.hpp:



## Namespaces

- namespace **QuantLib**

## 8.328 ql/PricingEngines/Vanilla/fdamericanengine.hpp File Reference

### 8.328.1 Detailed Description

Finite-differences American option engine.

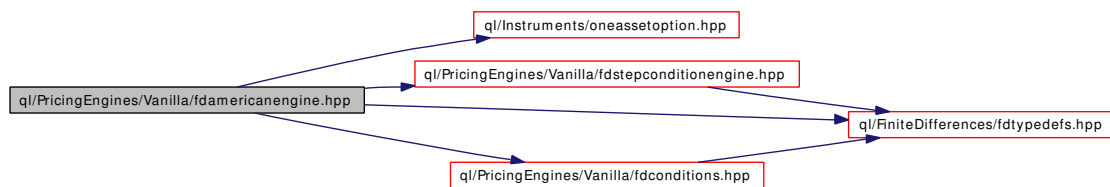
```
#include <ql/Instruments/oneassetoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdstepconditionengine.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdconditions.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

Include dependency graph for fdamericanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `FDEngineAdapter< FDAmericanCondition< FDStepConditionEngine >, OneAssetOption::engine >` [FDAmericanEngine](#)

*Finite-differences pricing engine for American one asset options.*

## 8.329 ql/PricingEngines/Vanilla/fdbermudanengine.hpp File Reference

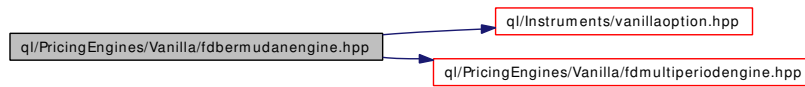
### 8.329.1 Detailed Description

finite-difference Bermudan engine

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp>
```

Include dependency graph for fdbermudanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FDBermudanEngine](#)  
*Finite-differences Bermudan engine.*

## 8.330 ql/PricingEngines/Vanilla/fdconditions.hpp File Reference

### 8.330.1 Detailed Description

Finite-difference templates to generate engines.

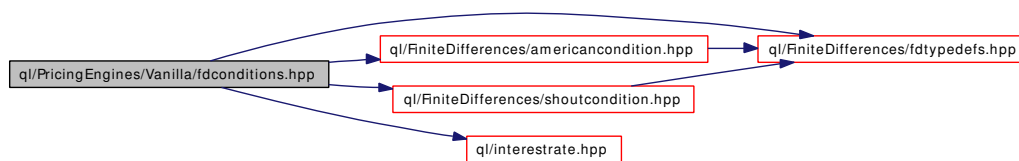
```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/FiniteDifferences/americancondition.hpp>
```

```
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

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

Include dependency graph for fdconditions.hpp:



### Namespaces

- namespace `QuantLib`

### Classes

- class `FDAmericanCondition`

## 8.331 ql/PricingEngines/Vanilla/fddividendamericanengine.hpp File Reference

### 8.331.1 Detailed Description

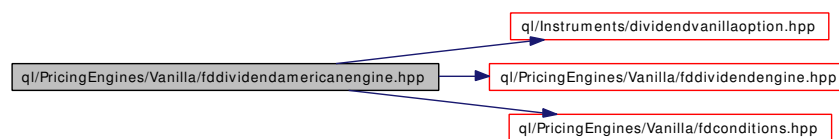
american engine with discrete deterministic dividends

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdconditions.hpp>
```

Include dependency graph for fddividendamericanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `FDEngineAdapter< FDAmericanCondition< FDDividendEngine >, DividendVanillaOption::engine >` [\*\*FDDividendAmericanEngine\*\*](#)  
*Finite-differences pricing engine for dividend American options.*
- typedef `FDEngineAdapter< FDAmericanCondition< FDDividendEngineMerton73 >, DividendVanillaOption::engine >` **FDDividendAmericanEngineMerton73**
- typedef `FDEngineAdapter< FDAmericanCondition< FDDividendEngineShiftScale >, DividendVanillaOption::engine >` **FDDividendAmericanEngineShiftScale**



## 8.332 ql/PricingEngines/Vanilla/fddividendengine.hpp File Reference

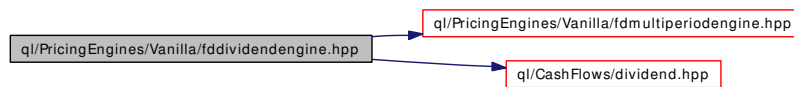
### 8.332.1 Detailed Description

base engine for option with dividends

```
#include <ql/PricingEngines/Vanilla/fdmultipleriodengine.hpp>
```

```
#include <ql/CashFlows/dividend.hpp>
```

Include dependency graph for fddividendengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FDDividendEngineMerton73](#)  
*Finite-differences pricing engine for dividend options using.*
- class [FDDividendEngineShiftScale](#)  
*Finite-differences pricing engine for dividend options using.*

### Typedefs

- typedef [FDDividendEngineMerton73](#) **FDDividendEngine**

## 8.333 ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp File Reference

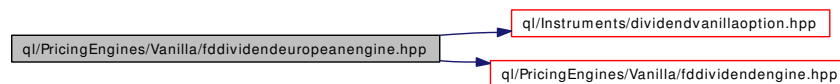
### 8.333.1 Detailed Description

finite-differences engine for European option with dividends

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>
```

Include dependency graph for fddividendeuropeanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef FEngineAdapter< FDDividendEngine, DividendVanillaOption::engine >  
[FDDividendEuropeanEngine](#)  
*Finite-differences pricing engine for dividend European options.*
- typedef FEngineAdapter< FDDividendEngineMerton73, DividendVanillaOption::engine >  
**FDDividendEuropeanEngineMerton73**
- typedef FEngineAdapter< FDDividendEngineShiftScale, DividendVanillaOption::engine >  
**FDDividendEuropeanEngineShiftScale**

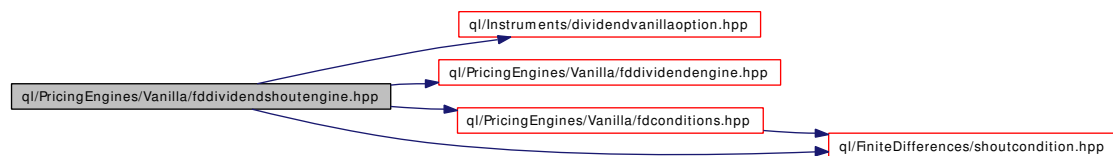
## 8.334 ql/PricingEngines/Vanilla/fddividendshoutengine.hpp File Reference

### 8.334.1 Detailed Description

base class for shout engine with dividends

```
#include <ql/Instruments/dividendvanillaoption.hpp>
#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>
#include <ql/PricingEngines/Vanilla/fdconditions.hpp>
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

Include dependency graph for fddividendshoutengine.hpp:



## Namespaces

- namespace **QuantLib**

## Typedefs

- typedef `FDEngineAdapter< FDS shoutCondition< FDDividendEngine >, DividendVanillaOption::engine >` **FDDividendShoutEngine**  
*Finite-differences shout engine with dividends.*
- typedef `FDEngineAdapter< FDS shoutCondition< FDDividendEngineMerton73 >, DividendVanillaOption::engine >` **FDDividendShoutEngineMerton73**
- typedef `FDEngineAdapter< FDS shoutCondition< FDDividendEngineShiftScale >, DividendVanillaOption::engine >` **FDDividendShoutEngineShiftScale**

## 8.335 ql/PricingEngines/Vanilla/fdeuropeanengine.hpp File Reference

### 8.335.1 Detailed Description

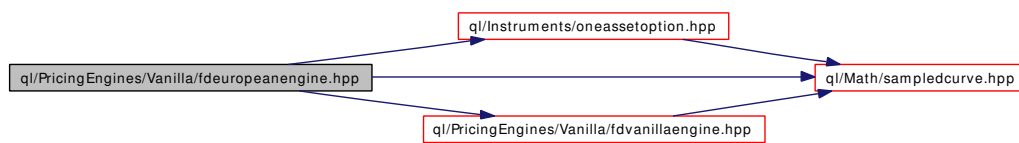
Finite-difference European engine.

```
#include <ql/Instruments/oneassetoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>
```

```
#include <ql/Math/sampledcurve.hpp>
```

Include dependency graph for fdeuropeanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **FDEuropeanEngine**

*Pricing engine for European options using finite-differences.*

## 8.336 ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp File Reference

### 8.336.1 Detailed Description

base engine for options with events happening at specific times

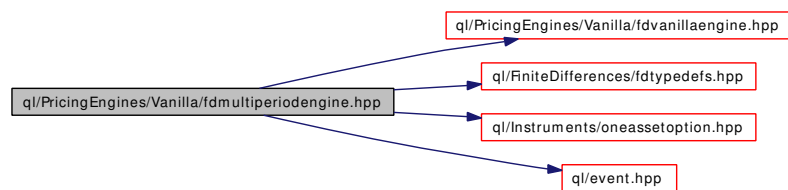
```
#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/Instruments/oneassetoption.hpp>
```

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

Include dependency graph for fdmultiperiodengine.hpp:



## Namespaces

- namespace **QuantLib**

## 8.337 ql/PricingEngines/Vanilla/fdshoutengine.hpp File Reference

### 8.337.1 Detailed Description

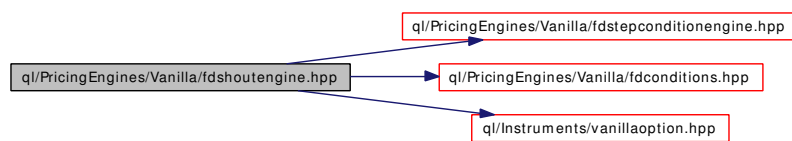
Finite-differences shout engine.

```
#include <ql/PricingEngines/Vanilla/fdstepconditionengine.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdconditions.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for fdshoutengine.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `FDEngineAdapter< FDShoutCondition< FDStepConditionEngine >, VanillaOption::engine >` [FDShoutEngine](#)

*Finite-differences pricing engine for shout vanilla options.*

## 8.338 ql/PricingEngines/Vanilla/fdstepconditionengine.hpp File Reference

### 8.338.1 Detailed Description

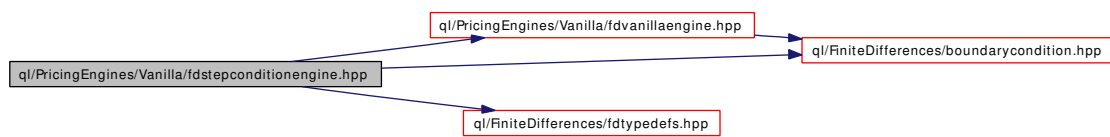
Finite-differences step-condition engine.

```
#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Include dependency graph for fdstepconditionengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FDStepConditionEngine](#)  
*Finite-differences pricing engine for American-style vanilla options.*

## 8.339 ql/PricingEngines/Vanilla/fdvanillaengine.hpp File Reference

### 8.339.1 Detailed Description

Finite-differences vanilla-option engine.

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

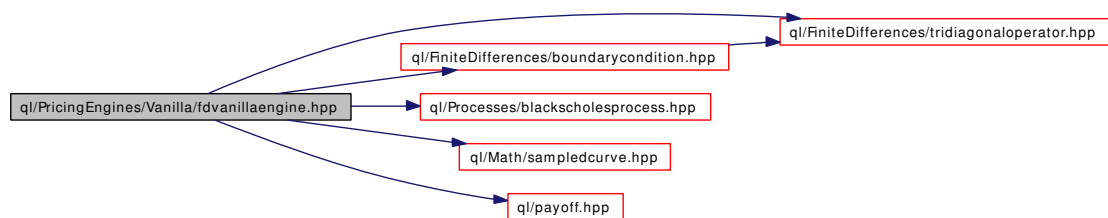
```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/Math/sampledcurve.hpp>
```

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

Include dependency graph for fdvanillaengine.hpp:



### Namespaces

- namespace **QuantLib**



## 8.340 ql/PricingEngines/Vanilla/integralengine.hpp File Reference

### 8.340.1 Detailed Description

Integral option engine.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for integralengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [IntegralEngine](#)

## 8.341 ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference

### 8.341.1 Detailed Description

Jump diffusion (Merton 1976) engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for jumpdiffusionengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [JumpDiffusionEngine](#)  
*Jump-diffusion engine for vanilla options.*

## 8.342 ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference

### 8.342.1 Detailed Description

Ju quadratic (1999) approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for juquadraticengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [JuQuadraticApproximationEngine](#)

## 8.343 ql/PricingEngines/Vanilla/mcamericanengine.hpp File Reference

### 8.343.1 Detailed Description

American Monte Carlo engine.

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

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

```
#include <ql/MonteCarlo/lsmbasissystem.hpp>
```

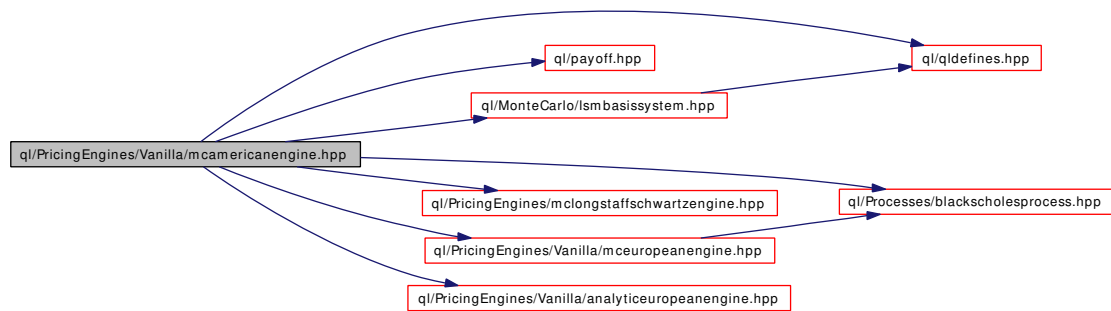
```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/PricingEngines/mclongstaffschwartzengine.hpp>
```

```
#include <ql/PricingEngines/Vanilla/mceuropeanengine.hpp>
```

```
#include <ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp>
```

Include dependency graph for mcamericanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCAmericanEngine](#)  
*American Monte Carlo engine.*
- class [MakeMCAmericanEngine](#)  
*Monte Carlo American engine factory.*

## 8.344 ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference

### 8.344.1 Detailed Description

digital option Monte Carlo engine

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

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

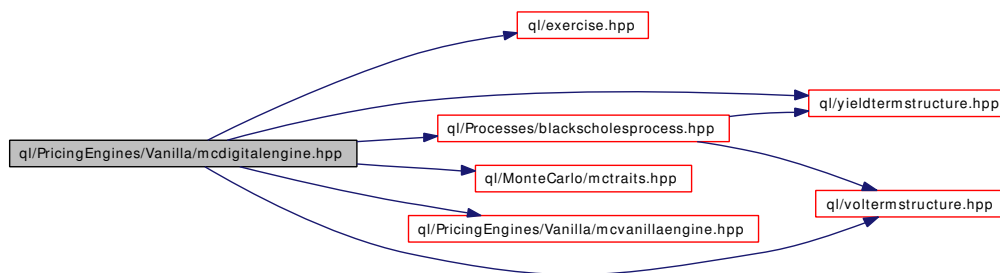
```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/MonteCarlo/mctraits.hpp>
```

```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for mcdigitalengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCDigitalEngine](#)  
*Pricing engine for digital options using Monte Carlo simulation.*
- class [MakeMCDigitalEngine](#)  
*Monte Carlo digital engine factory.*

## 8.345 ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference

### 8.345.1 Detailed Description

Monte Carlo European option engine.

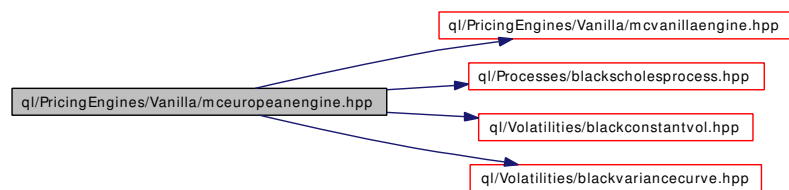
```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/Volatilities/blackconstantvol.hpp>
```

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for mceuropeanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCEuropeanEngine](#)  
*European option pricing engine using Monte Carlo simulation.*
- class [MakeMCEuropeanEngine](#)  
*Monte Carlo European engine factory.*

## 8.346 ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp File Reference

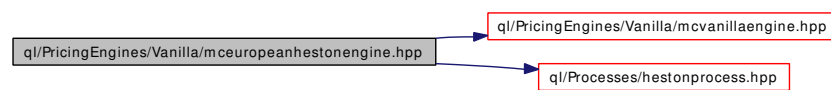
### 8.346.1 Detailed Description

Monte Carlo Heston-model engine for European options.

```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

```
#include <ql/Processes/hestonprocess.hpp>
```

Include dependency graph for mceuropeanhestonengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCEuropeanHestonEngine](#)  
*Monte Carlo Heston-model engine for European options.*
- class [MakeMCEuropeanHestonEngine](#)  
*Monte Carlo Heston European engine factory.*

## 8.347 ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference

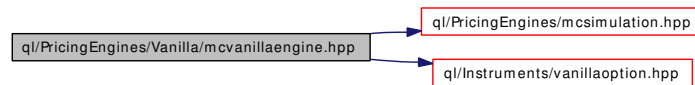
### 8.347.1 Detailed Description

Monte Carlo vanilla option engine.

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for mcvanillaengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCVanillaEngine](#)

*Pricing engine for vanilla options using Monte Carlo simulation.*



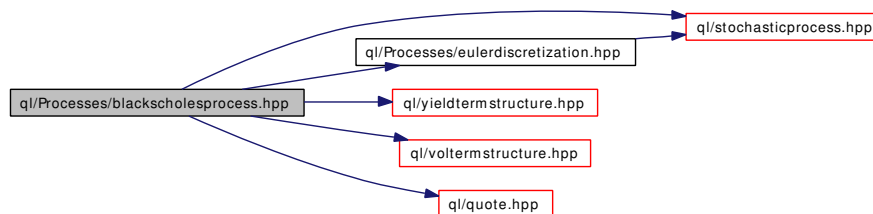
## 8.348 ql/Processes/blackscholesprocess.hpp File Reference

### 8.348.1 Detailed Description

Black-Scholes processes.

```
#include <ql/stochasticprocess.hpp>
#include <ql/Processes/eulerdiscretization.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/voltermstructure.hpp>
#include <ql/quote.hpp>
```

Include dependency graph for blackscholesprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GeneralizedBlackScholesProcess](#)  
*Generalized Black-Scholes stochastic process.*
- class [BlackScholesProcess](#)  
*Black-Scholes (1973) stochastic process.*
- class [BlackScholesMertonProcess](#)  
*Merton (1973) extension to the Black-Scholes stochastic process.*
- class [BlackProcess](#)  
*Black (1976) stochastic process.*
- class [GarmanKohlhagenProcess](#)  
*Garman-Kohlhagen (1983) stochastic process.*

### Typedefs

- typedef `BlackScholesProcess` [BlackScholes73Process](#)

## 8.349 ql/Processes/eulerdiscretization.hpp File Reference

### 8.349.1 Detailed Description

Euler discretization for stochastic processes.

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

Include dependency graph for eulerdiscretization.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [EulerDiscretization](#)  
*Euler discretization for stochastic processes.*

## 8.350 ql/Processes/forwardmeasureprocess.hpp File Reference

### 8.350.1 Detailed Description

forward-measure stochastic processes

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

Include dependency graph for forwardmeasureprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ForwardMeasureProcess](#)  
*forward-measure stochastic process*
- class [ForwardMeasureProcess1D](#)  
*forward-measure 1-D stochastic process*

## 8.351 ql/Processes/g2process.hpp File Reference

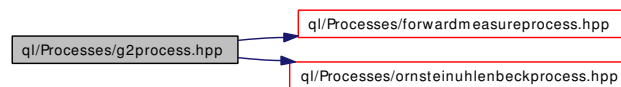
### 8.351.1 Detailed Description

G2 stochastic processes.

```
#include <ql/Processes/forwardmeasureprocess.hpp>
```

```
#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>
```

Include dependency graph for g2process.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [G2Process](#)  
*G2 stochastic process*
- class [G2ForwardProcess](#)  
*forward G2 stochastic process*

## 8.352 ql/Processes/geometricbrownianprocess.hpp File Reference

### 8.352.1 Detailed Description

Geometric Brownian-motion process.

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

Include dependency graph for geometricbrownianprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GeometricBrownianMotionProcess](#)  
*Geometric brownian-motion process.*

## 8.353 ql/Processes/hestonprocess.hpp File Reference

### 8.353.1 Detailed Description

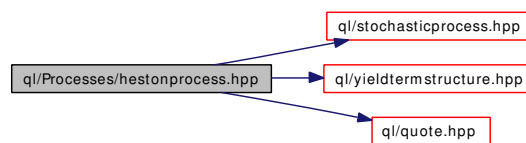
Heston stochastic process.

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

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

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

Include dependency graph for hestonprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [HestonProcess](#)  
*Square-root stochastic-volatility Heston process.*

## 8.354 ql/Processes/hullwhiteprocess.hpp File Reference

### 8.354.1 Detailed Description

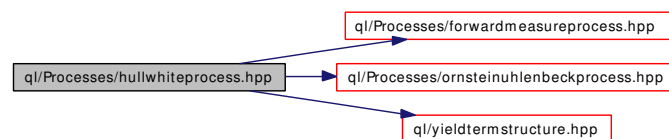
Hull-White stochastic processes.

```
#include <ql/Processes/forwardmeasureprocess.hpp>
```

```
#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>
```

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

Include dependency graph for hullwhiteprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [HullWhiteProcess](#)  
*Hull-White stochastic process.*
- class [HullWhiteForwardProcess](#)  
*forward Hull-White stochastic process*

## 8.355 ql/Processes/lfmcovarparam.hpp File Reference

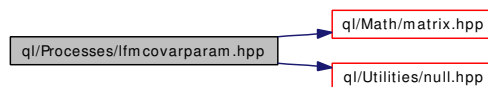
### 8.355.1 Detailed Description

volatility & correlation function for libor forward model process

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

Include dependency graph for lfmcovarparam.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LfmCovarianceParameterization](#)  
*libor market model parameterization*



## 8.356 ql/Processes/lfmhullwhiteparam.hpp File Reference

### 8.356.1 Detailed Description

libor market model parameterization based on Hull White

```
#include <ql/Processes/lfmprocess.hpp>
```

```
#include <ql/Processes/lfmcovarparam.hpp>
```

Include dependency graph for lfmhullwhiteparam.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LfmHullWhiteParameterization](#)

*libor market model parameterization based on Hull White paper*

## 8.357 ql/Processes/lfmprocess.hpp File Reference

### 8.357.1 Detailed Description

stochastic process of a libor forward model

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

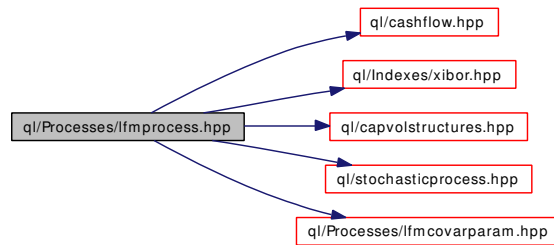
```
#include <ql/Indexes/xibor.hpp>
```

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

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

```
#include <ql/Processes/lfmcovarparam.hpp>
```

Include dependency graph for lfmprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LiborForwardModelProcess](#)  
*libor-forward-model process*

## 8.358 ql/Processes/merton76process.hpp File Reference

### 8.358.1 Detailed Description

Merton-76 process.

```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/Processes/eulerdiscretization.hpp>
```

Include dependency graph for merton76process.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Merton76Process](#)  
*Merton-76 jump-diffusion process.*

## 8.359 ql/Processes/ornsteinuhlenbeckprocess.hpp File Reference

### 8.359.1 Detailed Description

Ornstein-Uhlenbeck process.

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

Include dependency graph for ornsteinuhlenbeckprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [OrnsteinUhlenbeckProcess](#)  
*Ornstein-Uhlenbeck process class.*

## 8.360 ql/Processes/squarerootprocess.hpp File Reference

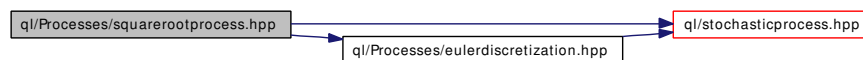
### 8.360.1 Detailed Description

square-root process

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

```
#include <ql/Processes/eulerdiscretization.hpp>
```

Include dependency graph for squarerootprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SquareRootProcess](#)  
*Square-root process class.*

## 8.361 ql/Processes/stochasticprocessarray.hpp File Reference

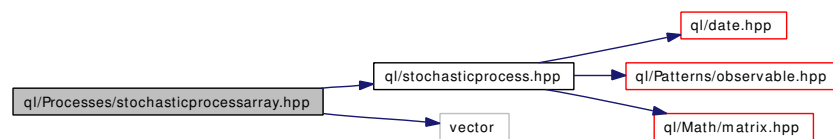
### 8.361.1 Detailed Description

Array of correlated 1-D stochastic processes.

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

```
#include <vector>
```

Include dependency graph for stochasticprocessarray.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [StochasticProcessArray](#)  
*Array of correlated 1-D stochastic processes*

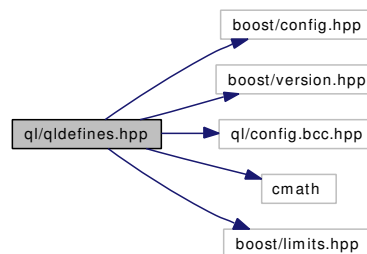
## 8.362 ql/qldefines.hpp File Reference

### 8.362.1 Detailed Description

Global definitions and compiler switches.

```
#include <boost/config.hpp>
#include <boost/version.hpp>
#include <ql/config.bcc.hpp>
#include <cmath>
#include <boost/limits.hpp>
```

Include dependency graph for qldefines.hpp:



### Defines

- **#define BOOST\_ENABLE\_ASSERT\_HANDLER**
- **#define QL\_INTEGER** int
- **#define QL\_BIG\_INTEGER** long
- **#define QL\_REAL** double
- **#define QL\_VERSION** "0.3.14"  
*version string*
- **#define QL\_HEX\_VERSION** 0x000314f0  
*version hexadecimal number*
- **#define QL\_LIB\_VERSION** "0\_3\_14"  
*version string for output lib name*
- **#define QL\_DUMMY\_RETURN(x)**  
*Is a dummy return statement required?*
- **#define QL\_IO\_INIT**  
*I/O initialization.*
- **#define QL\_MIN\_INTEGER** ((std::numeric\_limits<QL\_INTEGER>::min)())
- **#define QL\_MAX\_INTEGER** ((std::numeric\_limits<QL\_INTEGER>::max)())
- **#define QL\_MIN\_REAL** -((std::numeric\_limits<QL\_REAL>::max)())
- **#define QL\_MAX\_REAL** ((std::numeric\_limits<QL\_REAL>::max)())

- #define [QL\\_MIN\\_POSITIVE\\_REAL](#) ((std::numeric\_limits<QL\_REAL>::min)())
- #define [QL\\_EPSILON](#) ((std::numeric\_limits<QL\_REAL>::epsilon)())
- #define [QL\\_NULL\\_INTEGER](#) ((std::numeric\_limits<int>::max)())
- #define [QL\\_NULL\\_REAL](#) ((std::numeric\_limits<float>::max)())
- #define [QL\\_TYPENAME](#) typename
- #define [QL\\_FULL\\_ITERATOR\\_SUPPORT](#)



## 8.363 ql/quote.hpp File Reference

### 8.363.1 Detailed Description

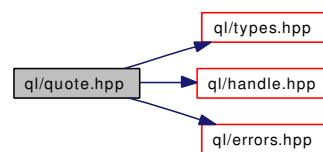
purely virtual base class for market observables

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

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

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

Include dependency graph for quote.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Quote](#)  
*purely virtual base class for market observables*
- class [SimpleQuote](#)  
*market element returning a stored value*
- class [DerivedQuote](#)  
*market element whose value depends on another market element*
- class [CompositeQuote](#)  
*market element whose value depends on two other market element*

## 8.364 ql/RandomNumbers/boxmullergaussianrng.hpp File Reference

### 8.364.1 Detailed Description

Box-Muller Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for boxmullergaussianrng.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BoxMullerGaussianRng](#)  
*Gaussian random number generator.*

## 8.365 ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference

### 8.365.1 Detailed Description

Central limit Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for centrallimitgaussianrng.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CLGaussianRng](#)  
*Gaussian random number generator.*

## 8.366 ql/RandomNumbers/faurersg.hpp File Reference

### 8.366.1 Detailed Description

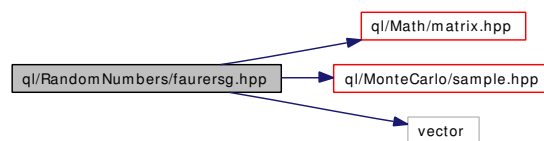
Faure low-discrepancy sequence generator.

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for faurersg.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FaureRsg](#)  
*Faure low-discrepancy sequence generator.*

## 8.367 ql/RandomNumbers/haltonrsg.hpp File Reference

### 8.367.1 Detailed Description

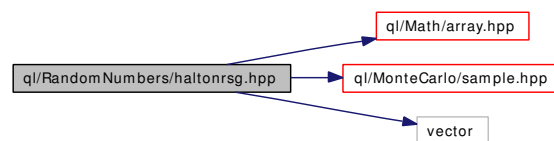
Halton low-discrepancy sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for haltonrsg.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [HaltonRsg](#)  
*Halton low-discrepancy sequence generator.*

## 8.368 ql/RandomNumbers/inversecumulativrng.hpp File Reference

### 8.368.1 Detailed Description

Inverse cumulative Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for inversecumulativrng.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InverseCumulativeRng](#)  
*Inverse cumulative random number generator.*

## 8.369 ql/RandomNumbers/inversecumulativergs.hpp File Reference

### 8.369.1 Detailed Description

Inverse cumulative random sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for inversecumulativergs.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InverseCumulativeRsg](#)  
*Inverse cumulative random sequence generator.*

## 8.370 ql/RandomNumbers/knuthuniformrng.hpp File Reference

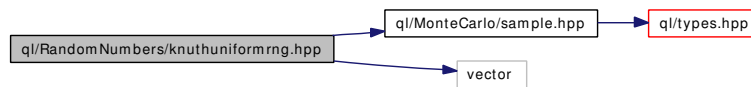
### 8.370.1 Detailed Description

Knuth uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for knuthuniformrng.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [KnuthUniformRng](#)  
*Uniform random number generator.*



## 8.371 ql/RandomNumbers/lecuyeruniformrng.hpp File Reference

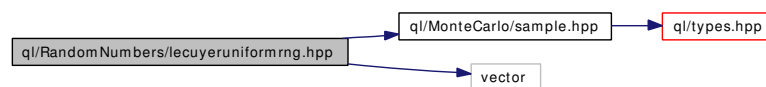
### 8.371.1 Detailed Description

L'Ecuyer uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for lecuyeruniformrng.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LecuyerUniformRng](#)  
*Uniform random number generator.*

## 8.372 ql/RandomNumbers/mt19937uniformrng.hpp File Reference

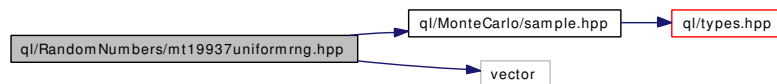
### 8.372.1 Detailed Description

Mersenne Twister uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for mt19937uniformrng.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MersenneTwisterUniformRng](#)  
*Uniform random number generator.*

## 8.373 ql/RandomNumbers/randomizedlds.hpp File Reference

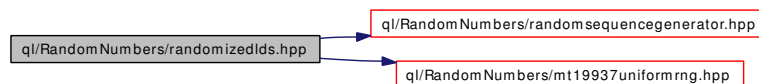
### 8.373.1 Detailed Description

Randomized low-discrepancy sequence.

```
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
```

```
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

Include dependency graph for randomizedlds.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [RandomizedLDS](#)  
*Randomized (random shift) low-discrepancy sequence.*

## 8.374 ql/RandomNumbers/randomsequencegenerator.hpp File Reference

### 8.374.1 Detailed Description

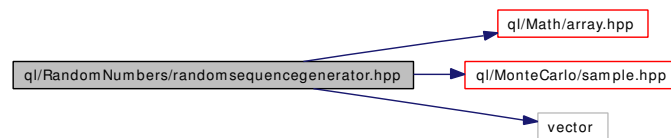
Random sequence generator based on a pseudo-random number generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for randomsequencegenerator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [RandomSequenceGenerator](#)

*Random sequence generator based on a pseudo-random number generator.*

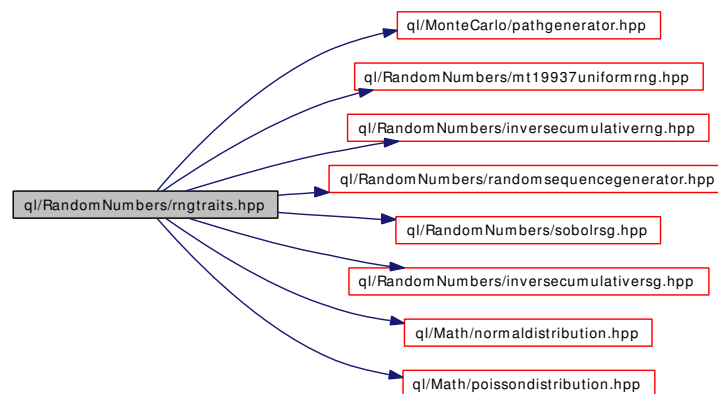
## 8.375 ql/RandomNumbers/rngtraits.hpp File Reference

### 8.375.1 Detailed Description

random-number generation policies

```
#include <ql/MonteCarlo/pathgenerator.hpp>
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
#include <ql/RandomNumbers/inversecumulativrng.hpp>
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
#include <ql/RandomNumbers/sobolrsg.hpp>
#include <ql/RandomNumbers/inversecumulativsg.hpp>
#include <ql/Math/normaldistribution.hpp>
#include <ql/Math/poissondistribution.hpp>
```

Include dependency graph for rngtraits.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef GenericPseudoRandom< MersenneTwisterUniformRng, InverseCumulativeNormal > [PseudoRandom](#)  
*default traits for pseudo-random number generation*
- typedef GenericPseudoRandom< MersenneTwisterUniformRng, InverseCumulativePoisson > [PoissonPseudoRandom](#)  
*traits for Poisson-distributed pseudo-random number generation*
- typedef GenericLowDiscrepancy< SobolRsg, InverseCumulativeNormal > [Low-Discrepancy](#)  
*default traits for low-discrepancy sequence generation*



## 8.376 ql/RandomNumbers/seedgenerator.hpp File Reference

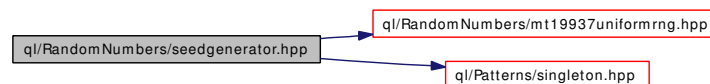
### 8.376.1 Detailed Description

Random seed generator.

```
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

Include dependency graph for seedgenerator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SeedGenerator](#)  
*Random seed generator.*

## 8.377 ql/RandomNumbers/sobolrsg.hpp File Reference

### 8.377.1 Detailed Description

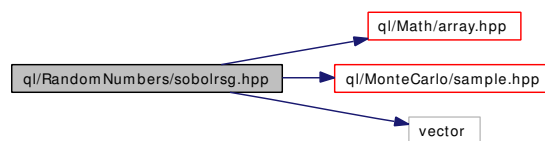
Sobol low-discrepancy sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for sobolrsg.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SobolRsg](#)  
*Sobol low-discrepancy sequence generator.*



## 8.378 ql/schedule.hpp File Reference

### 8.378.1 Detailed Description

date schedule

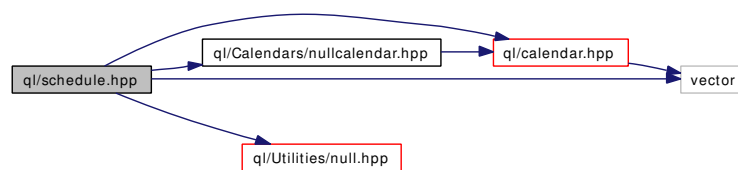
```
#include <ql/calendar.hpp>
```

```
#include <ql/Calendars/nullcalendar.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

```
#include <vector>
```

Include dependency graph for schedule.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [Schedule](#)  
*Payment schedule.*
- class [MakeSchedule](#)  
*helper class*

## 8.379 ql/settings.hpp File Reference

### 8.379.1 Detailed Description

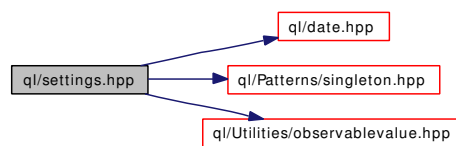
global repository for run-time library settings

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

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <ql/Utilities/observablevalue.hpp>
```

Include dependency graph for settings.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Settings](#)  
*global repository for run-time library settings*

### Functions

- `std::ostream & operator<< (std::ostream &out, const Settings::DateProxy &p)`

## 8.380 ql/ShortRateModels/calibrationhelper.hpp File Reference

### 8.380.1 Detailed Description

Calibration helper class.

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

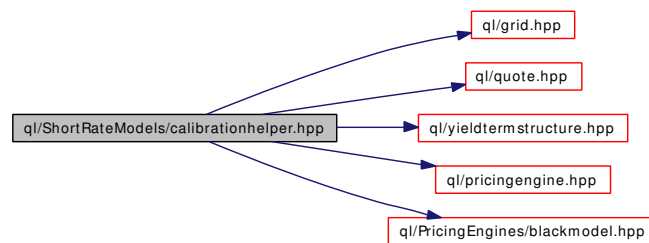
```
#include <ql/quote.hpp>
```

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

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

```
#include <ql/PricingEngines/blackmodel.hpp>
```

Include dependency graph for calibrationhelper.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **CalibrationHelper**  
*liquid market instrument used during calibration*

## 8.381 ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference

### 8.381.1 Detailed Description

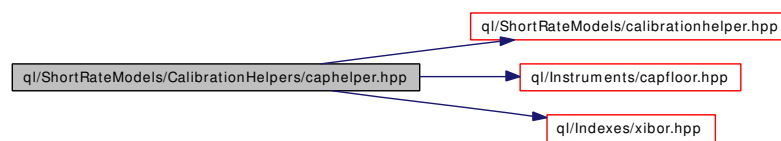
CapHelper calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for caphelper.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [CapHelper](#)  
*calibration helper for ATM cap*

## 8.382 ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp File Reference

### 8.382.1 Detailed Description

Heston-model calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for hestonmodelhelper.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [HestonModelHelper](#)  
*calibration helper for Heston model*

## 8.383 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference

### 8.383.1 Detailed Description

Swaption calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for swaptionhelper.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwaptionHelper](#)  
*calibration helper for ATM swaption*

## 8.384 ql/ShortRateModels/LiborMarketModels/lfmcovarproxy.hpp File Reference

### 8.384.1 Detailed Description

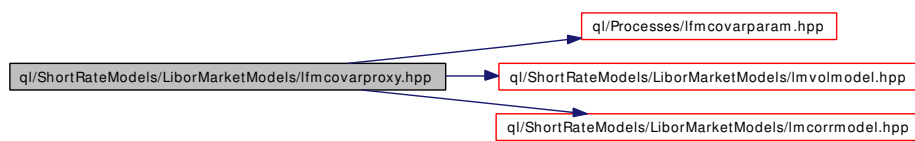
proxy for libor forward covariance parameterization

```
#include <ql/Processes/lfmcovarParams.hpp>
```

```
#include <ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp>
```

```
#include <ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp>
```

Include dependency graph for lfmcovarproxy.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [LfmCovarianceProxy](#)  
*proxy for a libor forward model covariance parameterization*

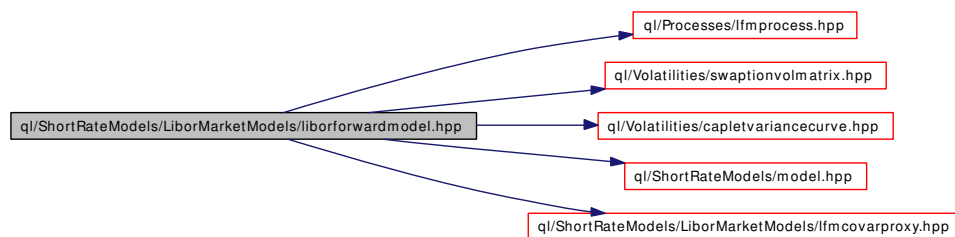
## 8.385 ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp File Reference

### 8.385.1 Detailed Description

libor forward model incl. exact cap pricing Rebonato formula to approximate swaption prices.

```
#include <ql/Processes/lfmprocess.hpp>
#include <ql/Volatilities/swaptionvolmatrix.hpp>
#include <ql/Volatilities/capletvariancecurve.hpp>
#include <ql/ShortRateModels/model.hpp>
#include <ql/ShortRateModels/LiborMarketModels/lfmcovarproxy.hpp>
```

Include dependency graph for liborforwardmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LiborForwardModel](#)  
*Libor Forward Model.*



## 8.386 ql/ShortRateModels/LiborMarketModels/lmconstwrappercorrmodel.h File Reference

### 8.386.1 Detailed Description

const wrapper for correlation model for libor market models

```
#include <ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp>
```

Include dependency graph for lmconstwrappercorrmodel.hpp:



### Namespaces

- namespace **QuantLib**

## 8.387 ql/ShortRateModels/LiborMarketModels/lmconstwrappervolmodel.hpp File Reference

### 8.387.1 Detailed Description

const wrapper for a volatility model for libor market models

```
#include <ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp>
```

Include dependency graph for lmconstwrappervolmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LmConstWrapperVolatilityModel](#)  
*caplet const volatility model*

## 8.388 ql/ShortRateModels/LiborMarketModels/Lmcorrmodel.hpp File Reference

### 8.388.1 Detailed Description

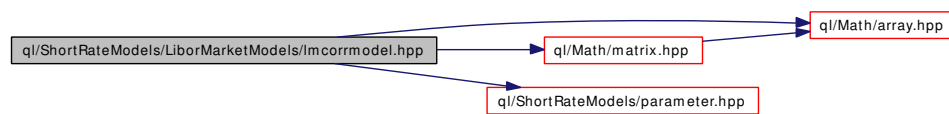
correlation model for libor market models

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/ShortRateModels/parameter.hpp>
```

Include dependency graph for `lmcorrmodel.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LmCorrelationModel](#)  
*libor forward correlation model*

## 8.389 ql/ShortRateModels/LiborMarketModels/lmexpcorrmodel.hpp File Reference

### 8.389.1 Detailed Description

exponential correlation model for libor market models

```
#include <ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp>
```

Include dependency graph for lmexpcorrmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LmExponentialCorrelationModel](#)  
*exponential correlation model*

## 8.390 ql/ShortRateModels/LiborMarketModels/lmextlinexpvolmodel.hpp File Reference

### 8.390.1 Detailed Description

volatility model for libor market models

```
#include <ql/ShortRateModels/LiborMarketModels/lmlinexpvolmodel.hpp>
```

Include dependency graph for lmextlinexpvolmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LmExtLinearExponentialVolModel](#)  
*extended linear exponential volatility model*

## 8.391 ql/ShortRateModels/LiborMarketModels/lmfixedvolmodel.hpp File Reference

### 8.391.1 Detailed Description

model of constant volatilities for libor market models

```
#include <ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp>
```

Include dependency graph for lmfixedvolmodel.hpp:



### Namespaces

- namespace **QuantLib**

## 8.392 ql/ShortRateModels/LiborMarketModels/lmlinexpcorrmodel.hpp File Reference

### 8.392.1 Detailed Description

exponential correlation model for libor market models

```
#include <ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp>
```

Include dependency graph for lmlinexpcorrmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LmLinearExponentialCorrelationModel](#)  
*linear exponential correlation model*

## 8.393 ql/ShortRateModels/LiborMarketModels/lmlinexpvolmodel.hpp File Reference

### 8.393.1 Detailed Description

volatility model for libor market models

```
#include <ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp>
```

Include dependency graph for lmlinexpvolmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LmLinearExponentialVolatilityModel](#)  
*linear exponential volatility model*



## 8.394 ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp File Reference

### 8.394.1 Detailed Description

volatility model for libor market models

```
#include <ql/ShortRateModels/parameter.hpp>
```

Include dependency graph for lmvolmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LmVolatilityModel](#)  
*caplet volatility model*

## 8.395 ql/ShortRateModels/model.hpp File Reference

### 8.395.1 Detailed Description

Abstract interest rate model class.

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

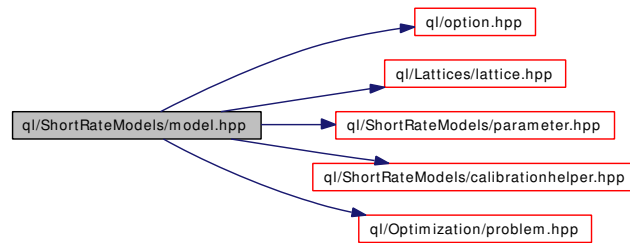
```
#include <ql/Lattices/lattice.hpp>
```

```
#include <ql/ShortRateModels/parameter.hpp>
```

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Optimization/problem.hpp>
```

Include dependency graph for model.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AffineModel](#)  
*Affine model class.*
- class [TermStructureConsistentModel](#)  
*Term-structure consistent model class.*
- class [CalibratedModel](#)  
*Calibrated model class.*
- class [ShortRateModel](#)  
*Abstract short-rate model class.*

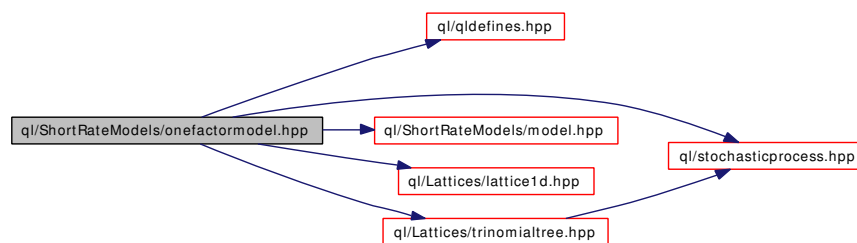
## 8.396 ql/ShortRateModels/onefactormodel.hpp File Reference

### 8.396.1 Detailed Description

Abstract one-factor interest rate model class.

```
#include <ql/qldefines.hpp>
#include <ql/stochasticprocess.hpp>
#include <ql/ShortRateModels/model.hpp>
#include <ql/Lattices/lattice1d.hpp>
#include <ql/Lattices/trinomialtree.hpp>
```

Include dependency graph for onefactormodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- 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.*

## 8.397 ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference

### 8.397.1 Detailed Description

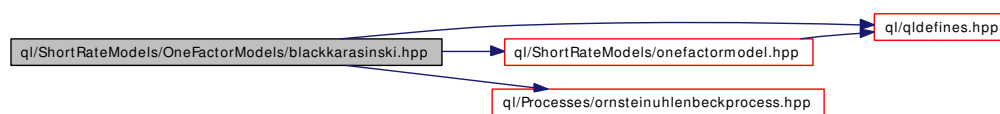
Black-Karasinski model.

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

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

```
#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>
```

Include dependency graph for blackkarasinski.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackKarasinski**  
*Standard Black-Karasinski model class.*
- class **BlackKarasinski::Dynamics**  
*Short-rate dynamics in the Black-Karasinski model.*

## 8.398 ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference

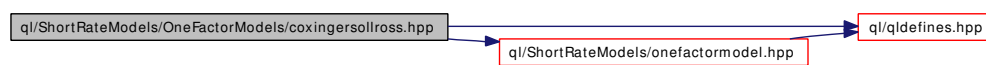
### 8.398.1 Detailed Description

Cox-Ingersoll-Ross model.

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

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for coxingersollross.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CoxIngersollRoss](#)  
*Cox-Ingersoll-Ross model class.*
- class [CoxIngersollRoss::Dynamics](#)  
*Dynamics of the short-rate under the Cox-Ingersoll-Ross model*

## 8.399 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference

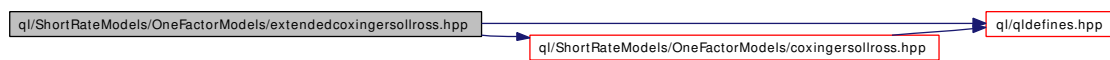
### 8.399.1 Detailed Description

Extended Cox-Ingersoll-Ross model.

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

```
#include <ql/ShortRateModels/OneFactorModels/coxingersollross.hpp>
```

Include dependency graph for extendedcoxingersollross.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- 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  $\varphi(t)$ .*

## 8.400 ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference

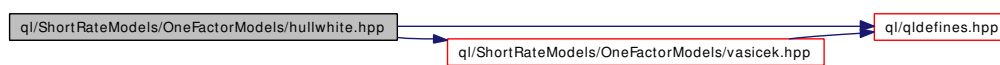
### 8.400.1 Detailed Description

Hull & White (HW) model.

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

```
#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>
```

Include dependency graph for hullwhite.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [HullWhite](#)  
*Single-factor Hull-White (extended Vasicek) model class.*
- class [HullWhite::Dynamics](#)  
*Short-rate dynamics in the Hull-White model.*
- class [HullWhite::FittingParameter](#)  
*Analytical term-structure fitting parameter  $\varphi(t)$ .*

## 8.401 ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference

### 8.401.1 Detailed Description

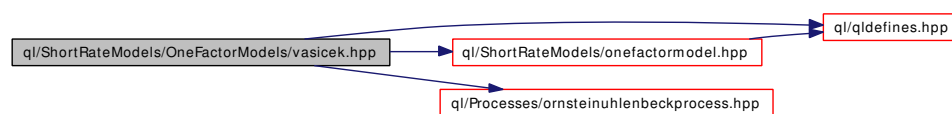
Vasicek model class.

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

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

```
#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>
```

Include dependency graph for vasicek.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [Vasicek](#)  
*Vasicek model class*
- class [Vasicek::Dynamics](#)  
*Short-rate dynamics in the Vasicek model.*



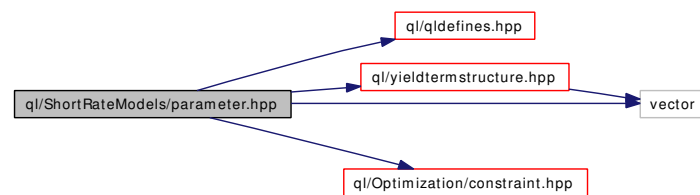
## 8.402 ql/ShortRateModels/parameter.hpp File Reference

### 8.402.1 Detailed Description

Model parameter classes.

```
#include <ql/qldefines.hpp>
#include <ql/yieldtermstructure.hpp>
#include <ql/Optimization/constraint.hpp>
#include <vector>
```

Include dependency graph for parameter.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- 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 which is always zero  $a(t) = 0$*
- class [PiecewiseConstantParameter](#)  
*Piecewise-constant parameter.*
- class [TermStructureFittingParameter](#)  
*Deterministic time-dependent parameter used for yield-curve fitting.*

## 8.403 ql/ShortRateModels/twofactormodel.hpp File Reference

### 8.403.1 Detailed Description

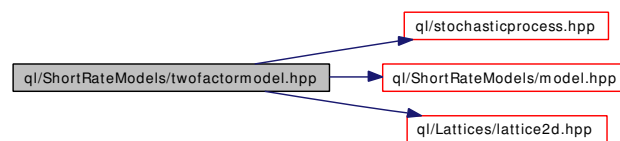
Abstract two-factor interest rate model class.

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

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/Lattices/lattice2d.hpp>
```

Include dependency graph for twofactormodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- 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.*

## 8.404 ql/ShortRateModels/TwoFactorModels/batesmodel.hpp File Reference

### 8.404.1 Detailed Description

extended versions of the Heston model

```
#include <ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp>
```

Include dependency graph for batesmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BatesModel](#)

## 8.405 ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference

### 8.405.1 Detailed Description

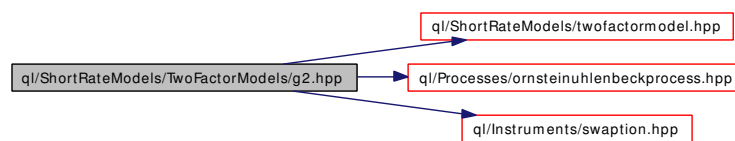
Two-factor additive Gaussian Model G2++.

```
#include <ql/ShortRateModels/twofactormodel.hpp>
```

```
#include <ql/Processes/ornsteinuhlenbeckprocess.hpp>
```

```
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for g2.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [G2](#)  
*Two-additive-factor gaussian model class.*
- class [G2::FittingParameter](#)  
*Analytical term-structure fitting parameter  $\varphi(t)$ .*

## 8.406 ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp File Reference

### 8.406.1 Detailed Description

Heston model for the stochastic volatility of an asset.

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/Processes/hestonprocess.hpp>
```

Include dependency graph for hestonmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [HestonModel](#)  
*Heston model for the stochastic volatility of an asset.*

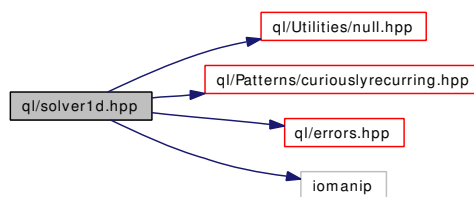
## 8.407 ql/solver1d.hpp File Reference

### 8.407.1 Detailed Description

Abstract 1-D solver class.

```
#include <ql/Utilities/null.hpp>
#include <ql/Patterns/curiouslyrecurring.hpp>
#include <ql/errors.hpp>
#include <iomanip>
```

Include dependency graph for solver1d.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Solver1D](#)  
*Base class for 1-D solvers.*

### Defines

- #define **MAX\_FUNCTION\_EVALUATIONS** 100

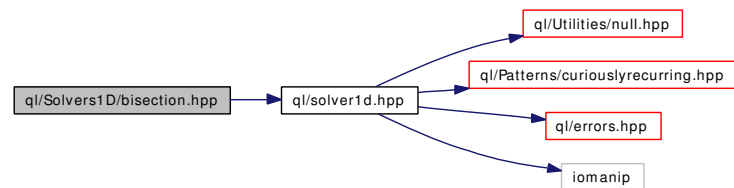
## 8.408 ql/Solvers1D/bisection.hpp File Reference

### 8.408.1 Detailed Description

bisection 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for bisection.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Bisection**  
*Bisection 1-D solver*

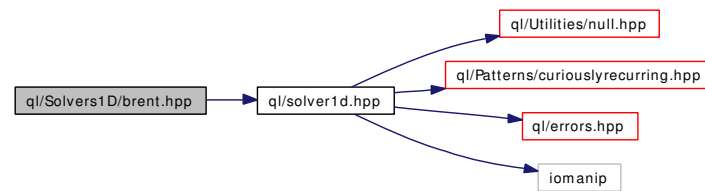
## 8.409 ql/Solvers1D/brent.hpp File Reference

### 8.409.1 Detailed Description

Brent 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for brent.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Brent**  
*Brent 1-D solver*



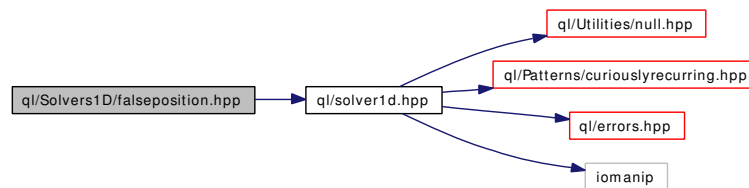
## 8.410 ql/Solvers1D/falseposition.hpp File Reference

### 8.410.1 Detailed Description

false-position 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for falseposition.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FalsePosition](#)  
*False position 1-D solver.*

## 8.411 ql/Solvers1D/newton.hpp File Reference

### 8.411.1 Detailed Description

Newton 1-D solver.

```
#include <ql/Solvers1D/newtonsafe.hpp>
```

Include dependency graph for newton.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Newton](#)  
*Newton 1-D solver*

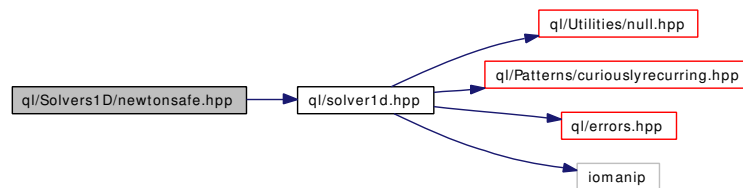
## 8.412 ql/Solvers1D/newtonsafe.hpp File Reference

### 8.412.1 Detailed Description

Safe (bracketed) Newton 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for newtonsafe.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [NewtonSafe](#)  
*safe Newton 1-D solver*

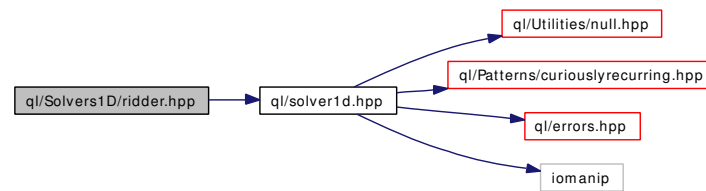
## 8.413 ql/Solvers1D/ridder.hpp File Reference

### 8.413.1 Detailed Description

Ridder 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for ridder.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Ridder](#)  
*Ridder 1-D solver*

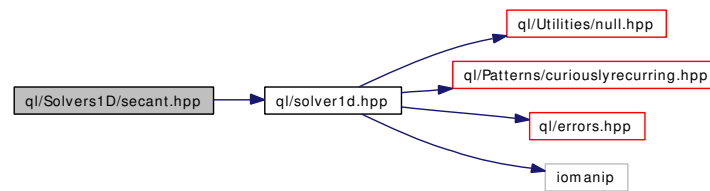
## 8.414 ql/Solvers1D/secant.hpp File Reference

### 8.414.1 Detailed Description

secant 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for secant.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Secant](#)  
*Secant 1-D solver*

## 8.415 ql/stochasticprocess.hpp File Reference

### 8.415.1 Detailed Description

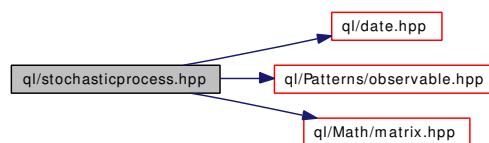
stochastic processes

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

```
#include <ql/Patterns/observable.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for stochasticprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [StochasticProcess](#)  
*multi-dimensional stochastic process class.*
- class [StochasticProcess::discretization](#)  
*discretization of a stochastic process over a given time interval*
- class [StochasticProcess1D](#)  
*1-dimensional stochastic process*
- class [StochasticProcess1D::discretization](#)  
*discretization of a 1-D stochastic process*

## 8.416 ql/swaptionvolstructure.hpp File Reference

### 8.416.1 Detailed Description

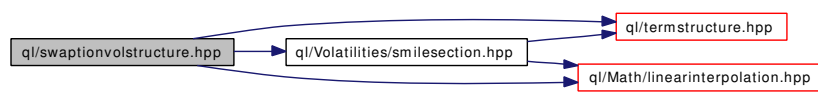
Swaption volatility structure.

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

```
#include <ql/Math/linearinterpolation.hpp>
```

```
#include <ql/Volatilities/smilesection.hpp>
```

Include dependency graph for swaptionvolstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwaptionVolatilityStructure](#)  
*Swaption-volatility structure*

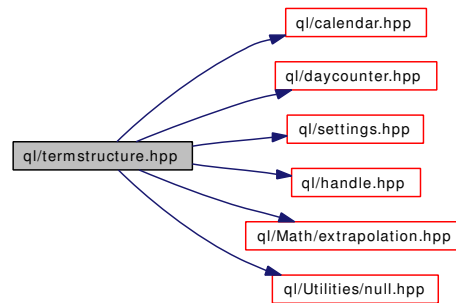
## 8.417 ql/termstructure.hpp File Reference

### 8.417.1 Detailed Description

base class for term structures

```
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/settings.hpp>
#include <ql/handle.hpp>
#include <ql/Math/extrapolation.hpp>
#include <ql/Utilities/null.hpp>
```

Include dependency graph for termstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TermStructure](#)  
*Basic term-structure functionality.*



## 8.418 ql/TermStructures/bondhelpers.hpp File Reference

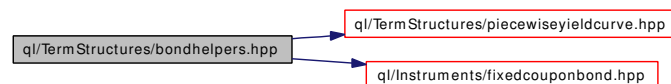
### 8.418.1 Detailed Description

bond rate helpers

```
#include <ql/TermStructures/piecewiseyieldcurve.hpp>
```

```
#include <ql/Instruments/fixedcouponbond.hpp>
```

Include dependency graph for bondhelpers.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **FixedCouponBondHelper**  
*fixed-coupon bond helper*

## 8.419 ql/TermStructures/bootstraptraits.hpp File Reference

### 8.419.1 Detailed Description

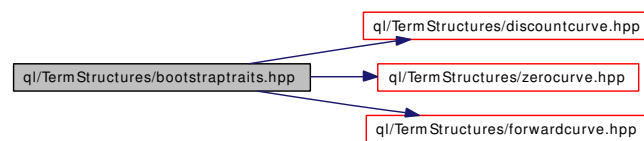
bootstrap traits

```
#include <ql/TermStructures/discountcurve.hpp>
```

```
#include <ql/TermStructures/zerocurve.hpp>
```

```
#include <ql/TermStructures/forwardcurve.hpp>
```

Include dependency graph for bootstraptraits.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct **Discount**  
*Discount-curve traits.*
- struct **ZeroYield**  
*Zero-curve traits.*
- struct **ForwardRate**  
*Forward-curve traits.*

## 8.420 ql/TermStructures/compoundforward.hpp File Reference

### 8.420.1 Detailed Description

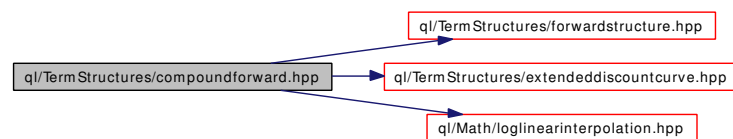
compounded forward term structure

```
#include <ql/TermStructures/forwardstructure.hpp>
```

```
#include <ql/TermStructures/extendeddiscountcurve.hpp>
```

```
#include <ql/Math/loglinearinterpolation.hpp>
```

Include dependency graph for compoundforward.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **CompoundForward**  
*compound-forward structure*

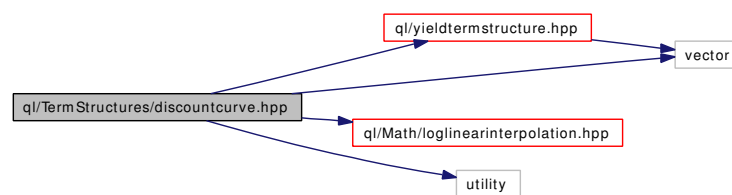
## 8.421 ql/TermStructures/discountcurve.hpp File Reference

### 8.421.1 Detailed Description

interpolated discount factor structure

```
#include <ql/yieldtermstructure.hpp>
#include <ql/Math/loglinearinterpolation.hpp>
#include <vector>
#include <utility>
```

Include dependency graph for discountcurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterpolatedDiscountCurve](#)  
*Term structure based on interpolation of discount factors.*

### Typedefs

- typedef `InterpolatedDiscountCurve< LogLinear >` [DiscountCurve](#)  
*Term structure based on log-linear interpolation of discount factors.*

## 8.422 ql/TermStructures/drifttermstructure.hpp File Reference

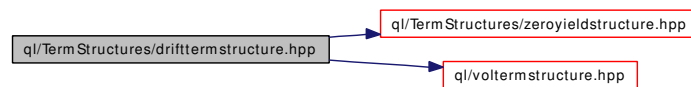
### 8.422.1 Detailed Description

Drift term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

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

Include dependency graph for drifttermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DriftTermStructure](#)

*Drift term structure.*

## 8.423 ql/TermStructures/extendeddiscountcurve.hpp File Reference

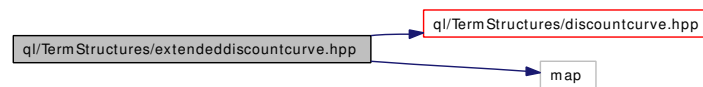
### 8.423.1 Detailed Description

discount factor structure with detailed compound-forward calculation

```
#include <ql/TermStructures/discountcurve.hpp>
```

```
#include <map>
```

Include dependency graph for extendeddiscountcurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ExtendedDiscountCurve](#)

*Term structure based on loglinear interpolation of discount factors.*

## 8.424 ql/TermStructures/flatforward.hpp File Reference

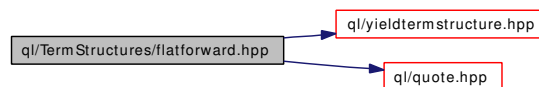
### 8.424.1 Detailed Description

flat forward rate term structure

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

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

Include dependency graph for flatforward.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FlatForward](#)  
*Flat interest-rate curve.*

## 8.425 ql/TermStructures/forwardcurve.hpp File Reference

### 8.425.1 Detailed Description

interpolated forward-rate structure

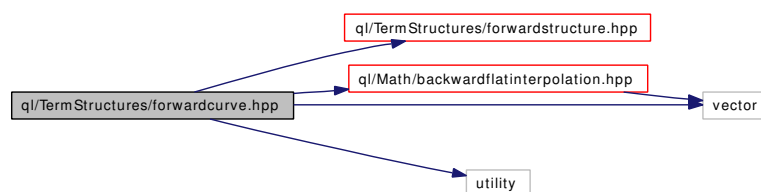
```
#include <ql/TermStructures/forwardstructure.hpp>
```

```
#include <ql/Math/backwardflatinterpolation.hpp>
```

```
#include <vector>
```

```
#include <utility>
```

Include dependency graph for forwardcurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterpolatedForwardCurve](#)  
*Term structure based on interpolation of forward rates.*

### Typedefs

- typedef `InterpolatedForwardCurve< BackwardFlat >` [ForwardCurve](#)  
*Term structure based on flat interpolation of forward rates.*



## 8.426 ql/TermStructures/forwardspreadedtermstructure.hpp File Reference

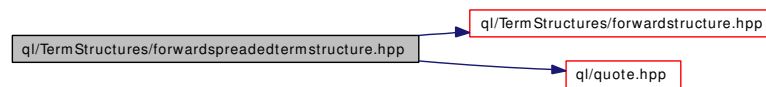
### 8.426.1 Detailed Description

Forward-spreaded term structure.

```
#include <ql/TermStructures/forwardstructure.hpp>
```

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

Include dependency graph for forwardspreadedtermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ForwardSpreadedTermStructure](#)

*Term structure with added spread on the instantaneous forward rate.*

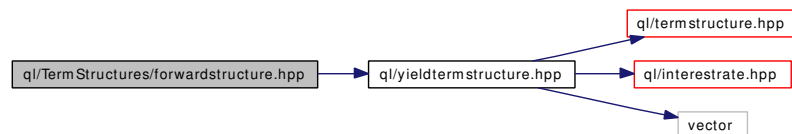
## 8.427 ql/TermStructures/forwardstructure.hpp File Reference

### 8.427.1 Detailed Description

Forward-based yield term structure.

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

Include dependency graph for forwardstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ForwardRateStructure](#)  
*Forward* rate term structure.

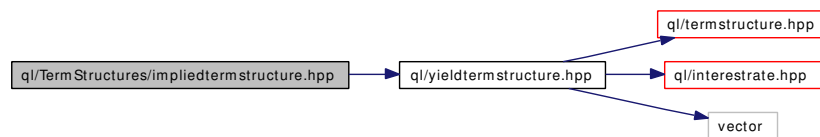
## 8.428 ql/TermStructures/IMPLIEDTERMSTRUCTURE.hpp File Reference

### 8.428.1 Detailed Description

Implied term structure.

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

Include dependency graph for IMPLIEDTERMSTRUCTURE.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ImpliedTermStructure](#)  
*Implied term structure at a given date in the future.*

## 8.429 ql/TermStructures/piecewiseflatforward.hpp File Reference

### 8.429.1 Detailed Description

piecewise flat forward term structure

```
#include <ql/TermStructures/piecewiseyieldcurve.hpp>
```

Include dependency graph for piecewiseflatforward.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `PiecewiseYieldCurve< Discount, LogLinear >` [PiecewiseFlatForward](#)  
*Piecewise flat-forward term structure.*

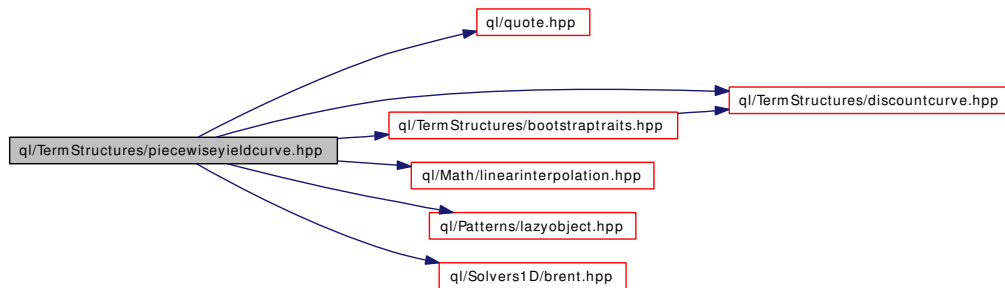
## 8.430 ql/TermStructures/pieewiseyieldcurve.hpp File Reference

### 8.430.1 Detailed Description

piecewise-interpolated term structure

```
#include <ql/quote.hpp>
#include <ql/TermStructures/discountcurve.hpp>
#include <ql/TermStructures/bootstraptraits.hpp>
#include <ql/Math/linearinterpolation.hpp>
#include <ql/Patterns/lazyobject.hpp>
#include <ql/Solvers1D/brent.hpp>
```

Include dependency graph for pieewiseyieldcurve.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class **RateHelper**  
*Base helper class for yield-curve bootstrapping.*
- class **PiecewiseYieldCurve**  
*Piecewise yield term structure.*

## 8.431 ql/TermStructures/piecewisezerospreadedtermstructure.hpp File Reference

### 8.431.1 Detailed Description

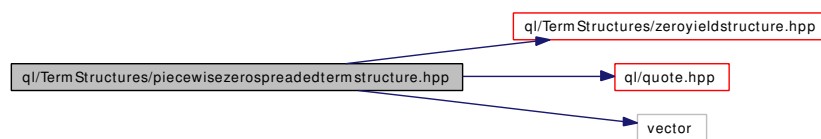
Piecewise-zero-spreaded term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

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

```
#include <vector>
```

Include dependency graph for piecewisezerospreadedtermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [PiecewiseZeroSpreadedTermStructure](#)  
*Term structure with an added vector of spreads on the zero-yield rate.*

## 8.432 ql/TermStructures/quantotermstructure.hpp File Reference

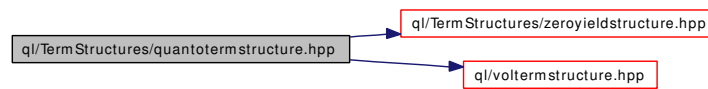
### 8.432.1 Detailed Description

Quanto term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

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

Include dependency graph for quantotermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [QuantoTermStructure](#)  
*Quanto term structure.*

## 8.433 ql/TermStructures/ratehelpers.hpp File Reference

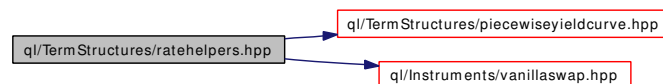
### 8.433.1 Detailed Description

deposit, FRA, futures, and swap rate helpers

```
#include <ql/TermStructures/piecewiseyieldcurve.hpp>
```

```
#include <ql/Instruments/vanillaswap.hpp>
```

Include dependency graph for ratehelpers.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FuturesRateHelper](#)  
*Rate helper for bootstrapping over interest-rate futures prices.*
- class [RelativeDateRateHelper](#)  
*Rate helper with date schedule relative to the global evaluation date.*
- class [DepositRateHelper](#)  
*Rate helper for bootstrapping over deposit rates.*
- class [FraRateHelper](#)  
*Rate helper for bootstrapping over FRA rates.*
- class [SwapRateHelper](#)  
*Rate helper for bootstrapping over swap rates.*



## 8.434 ql/TermStructures/zerocurve.hpp File Reference

### 8.434.1 Detailed Description

interpolated zero-rates structure

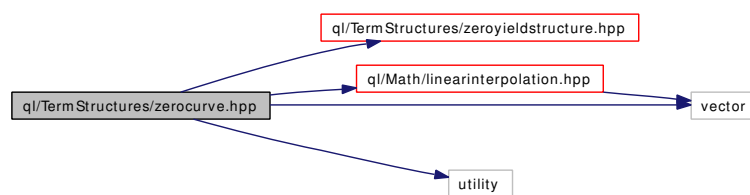
```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

```
#include <ql/Math/linearinterpolation.hpp>
```

```
#include <vector>
```

```
#include <utility>
```

Include dependency graph for zerocurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterpolatedZeroCurve](#)  
*Term structure based on interpolation of zero yields.*

### Typedefs

- typedef `InterpolatedZeroCurve< Linear >` [ZeroCurve](#)  
*Term structure based on linear interpolation of zero yields.*

## 8.435 ql/TermStructures/zerospreadedtermstructure.hpp File Reference

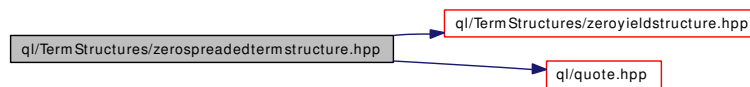
### 8.435.1 Detailed Description

Zero spreaded term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

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

Include dependency graph for zerospreadedtermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ZeroSpreadedTermStructure](#)  
*Term structure with an added spread on the zero yield rate.*

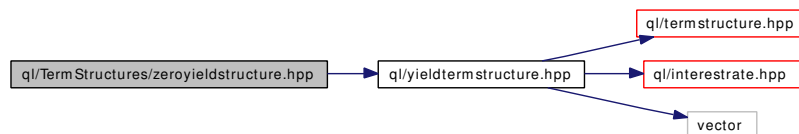
## 8.436 ql/TermStructures/zeroyieldstructure.hpp File Reference

### 8.436.1 Detailed Description

Zero-yield based term structure.

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

Include dependency graph for zeroyieldstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ZeroYieldStructure](#)  
*Zero-yield term structure.*

## 8.437 ql/timegrid.hpp File Reference

### 8.437.1 Detailed Description

discrete time grid

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

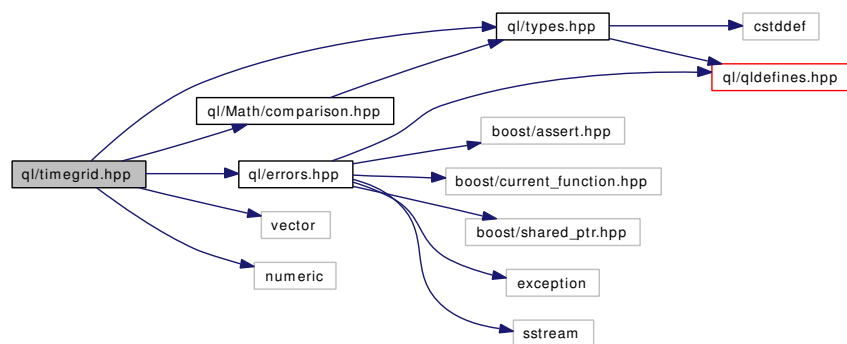
```
#include <ql/errors.hpp>
```

```
#include <ql/Math/comparison.hpp>
```

```
#include <vector>
```

```
#include <numeric>
```

Include dependency graph for timegrid.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TimeGrid](#)  
*time grid class*

## 8.438 ql/timeseries.hpp File Reference

### 8.438.1 Detailed Description

timeseries class

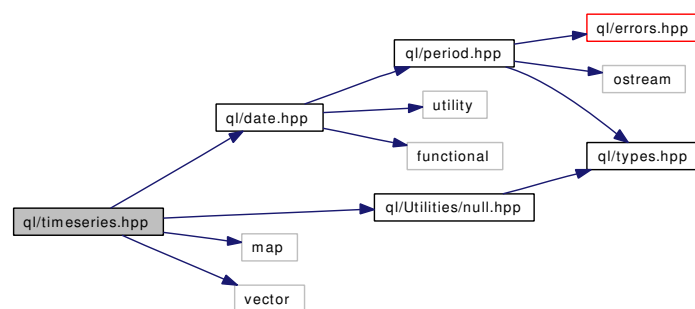
```
#include <ql/date.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

```
#include <map>
```

```
#include <vector>
```

Include dependency graph for timeseries.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TimeSeries](#)  
*Container for historical data.*

## 8.439 ql/types.hpp File Reference

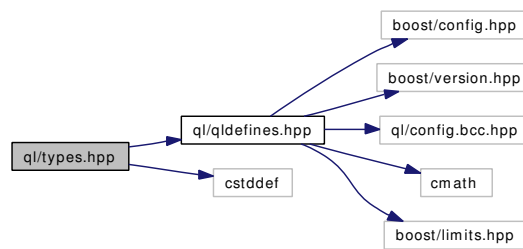
### 8.439.1 Detailed Description

Custom types.

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

```
#include <cstdint>
```

Include dependency graph for types.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef QL\_INTEGER **Integer**  
*integer number*
- typedef QL\_BIG\_INTEGER **BigInteger**  
*large integer number*
- typedef unsigned QL\_INTEGER **Natural**  
*positive integer*
- typedef unsigned QL\_BIG\_INTEGER **BigNatural**  
*large positive integer*
- typedef QL\_REAL **Real**  
*real number*
- typedef Real **Decimal**  
*decimal number*
- typedef std::size\_t **Size**  
*size of a container*
- typedef Real **Time**  
*continuous quantity with 1-year units*

- typedef Real [DiscountFactor](#)  
*discount factor between dates*
- typedef Real [Rate](#)  
*interest rates*
- typedef Real [Spread](#)  
*spreads on interest rates*
- typedef Real [Volatility](#)  
*volatility*

## 8.440 ql/Utilities/clone.hpp File Reference

### 8.440.1 Detailed Description

cloning proxy to an underlying object

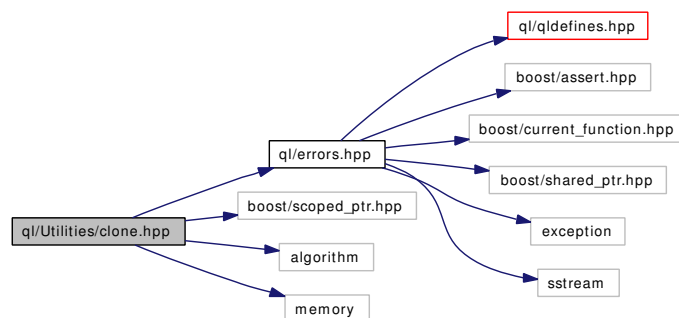
```
#include <ql/errors.hpp>
```

```
#include <boost/scoped_ptr.hpp>
```

```
#include <algorithm>
```

```
#include <memory>
```

Include dependency graph for clone.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [Clone](#)  
*cloning proxy to an underlying object*



## 8.441 ql/Utilities/dataformatters.hpp File Reference

### 8.441.1 Detailed Description

output manipulators

```
#include <ql/Utilities/null.hpp>
```

```
#include <ostream>
```

Include dependency graph for dataformatters.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**
- namespace **QuantLib::io**

### Functions

- `template<typename T> std::ostream & operator<< (std::ostream &, const null_checker< T > &)`
- `std::ostream & operator<< (std::ostream &, const ordinal_holder &)`
- `template<typename T> std::ostream & operator<< (std::ostream &, const power_of_two_holder< T > &)`
- `std::ostream & operator<< (std::ostream &, const percent_holder &)`
- `template<typename T> detail::null_checker< T > checknull (T)`  
*check for nulls before output*
- `detail::ordinal_holder ordinal (Size)`  
*outputs naturals as 1st, 2nd, 3rd...*
- `template<typename T> detail::power_of_two_holder< T > power_of_two (T)`  
*output integers as powers of two*
- `detail::percent_holder percent (Real)`  
*output reals as percentages*
- `detail::percent_holder rate (Rate)`  
*output rates and spreads as percentages*
- `detail::percent_holder volatility (Volatility)`  
*output volatilities as percentages*

## 8.442 ql/Utilities/dataparsers.hpp File Reference

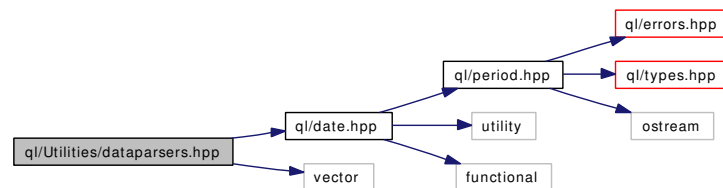
### 8.442.1 Detailed Description

Classes used to parse data for input.

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

```
#include <vector>
```

Include dependency graph for dataparsers.hpp:



## Namespaces

- namespace **QuantLib**

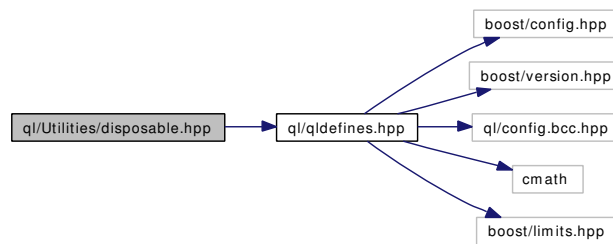
## 8.443 ql/Utilities/disposable.hpp File Reference

### 8.443.1 Detailed Description

generic disposable object with move semantics

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

Include dependency graph for disposable.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Disposable](#)  
*generic disposable object with move semantics*

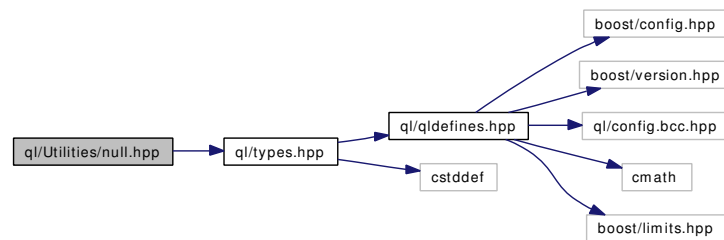
## 8.444 ql/Utilities/null.hpp File Reference

### 8.444.1 Detailed Description

null values

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

Include dependency graph for null.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Null](#)  
*template class providing a null value for a given type.*

## 8.445 ql/Utilities/observablevalue.hpp File Reference

### 8.445.1 Detailed Description

observable and assignable proxy to concrete value

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for observablevalue.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **ObservableValue**  
*observable and assignable proxy to concrete value*

## 8.446 ql/Utilities/steppingiterator.hpp File Reference

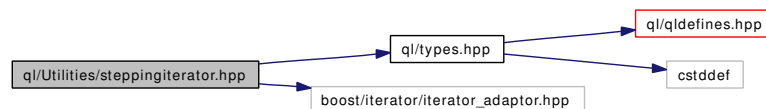
### 8.446.1 Detailed Description

Iterator advancing in constant steps.

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

```
#include <boost/iterator/iterator_adaptor.hpp>
```

Include dependency graph for steppingiterator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class `step_iterator`  
*Iterator advancing in constant steps.*

## 8.447 ql/Utilities/strings.hpp File Reference

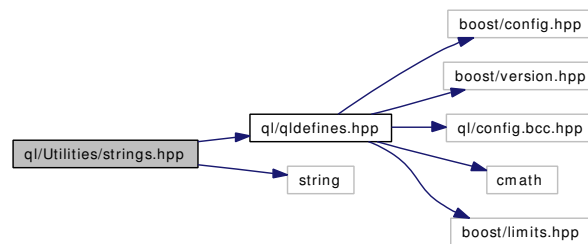
### 8.447.1 Detailed Description

string utilities

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

```
#include <string>
```

Include dependency graph for strings.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- `std::string lowercase` (const std::string &)
- `std::string uppercase` (const std::string &)

## 8.448 ql/Utilities/tracing.hpp File Reference

### 8.448.1 Detailed Description

tracing facilities

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

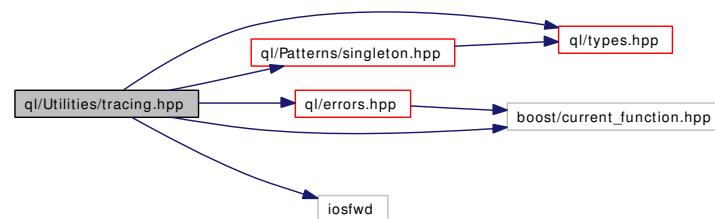
```
#include <ql/errors.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <boost/current_function.hpp>
```

```
#include <iosfwd>
```

Include dependency graph for tracing.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Defines

- `#define QL_DEFAULT_TRACER`
- `#define QL_TRACE_ENABLE`  
*enable tracing*
- `#define QL_TRACE_DISABLE`  
*disable tracing*
- `#define QL_TRACE_ON(out)`  
*set tracing stream*
- `#define QL_TRACE(message)`  
*output tracing information*
- `#define QL_TRACE_ENTER_FUNCTION`  
*output tracing information*
- `#define QL_TRACE_EXIT_FUNCTION`  
*output tracing information*



- #define [QL\\_TRACE\\_LOCATION](#)  
*output tracing information*
- #define [QL\\_TRACE\\_VARIABLE](#)(variable)  
*output tracing information*

## 8.449 ql/Volatilities/blackconstantvol.hpp File Reference

### 8.449.1 Detailed Description

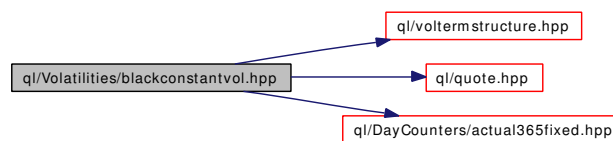
Black constant volatility, no time dependence, no strike dependence.

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

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

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for blackconstantvol.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BlackConstantVol](#)  
*Constant Black volatility, no time-strike dependence.*

## 8.450 ql/Volatilities/blackvariancecurve.hpp File Reference

### 8.450.1 Detailed Description

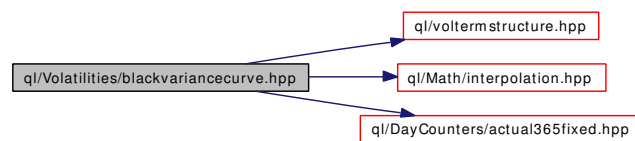
Black volatility curve modelled as variance curve.

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

```
#include <ql/Math/interpolation.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for blackvariancecurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BlackVarianceCurve](#)  
*Black volatility curve modelled as variance curve.*

## 8.451 ql/Volatilities/blackvariancesurface.hpp File Reference

### 8.451.1 Detailed Description

Black volatility surface modelled as variance surface.

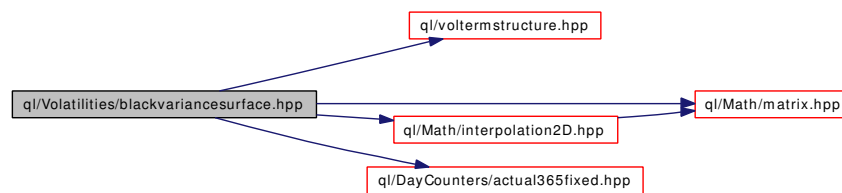
```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Math/interpolation2D.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for blackvariancesurface.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackVarianceSurface**  
*Black volatility surface modelled as variance surface.*

## 8.452 ql/Volatilities/capflatvolvector.hpp File Reference

### 8.452.1 Detailed Description

Cap/floor at-the-money flat volatility vector.

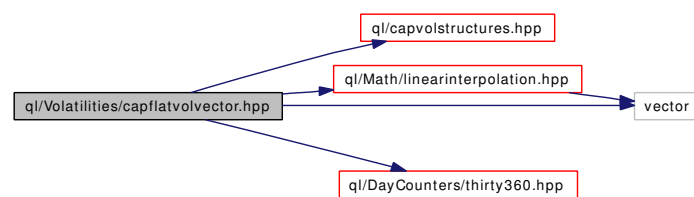
```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/Math/linearinterpolation.hpp>
```

```
#include <ql/DayCounters/thirty360.hpp>
```

```
#include <vector>
```

Include dependency graph for capflatvolvector.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CapVolatilityVector](#)  
*Cap/floor at-the-money term-volatility vector.*

## 8.453 ql/Volatilities/capletconstantvol.hpp File Reference

### 8.453.1 Detailed Description

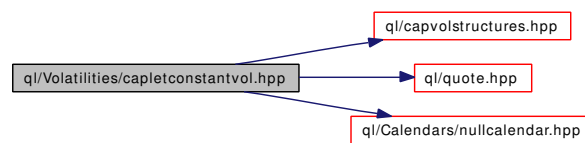
Constant caplet volatility.

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

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

```
#include <ql/Calendars/nullcalendar.hpp>
```

Include dependency graph for capletconstantvol.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CapletConstantVolatility](#)  
*Constant caplet volatility, no time-strike dependence.*

## 8.454 ql/Volatilities/capletvariancecurve.hpp File Reference

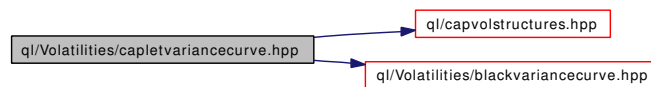
### 8.454.1 Detailed Description

caplet variance curve

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

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for capletvariancecurve.hpp:



### Namespaces

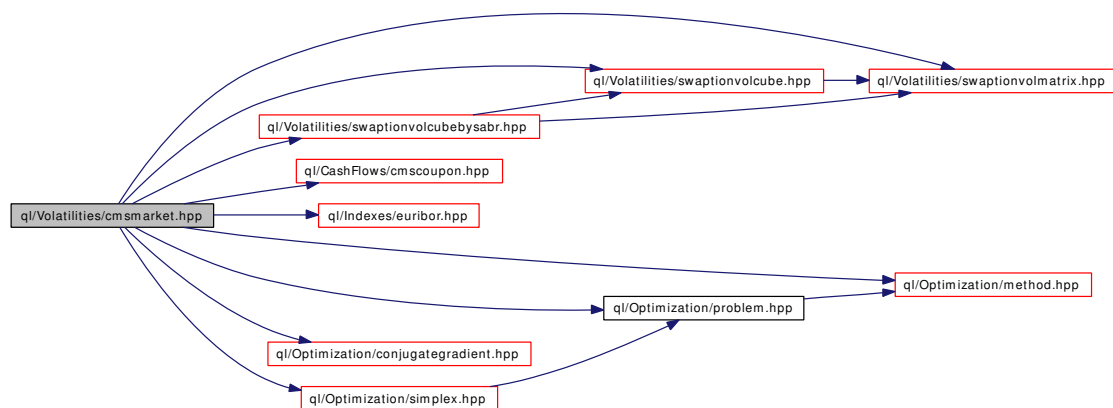
- namespace **QuantLib**

## 8.455 ql/Volatilities/cmsmarket.hpp File Reference

### 8.455.1 Detailed Description

```
#include <ql/Volatilities/swaptionvolmatrix.hpp>
#include <ql/Volatilities/swaptionvolcube.hpp>
#include <ql/Volatilities/swaptionvolcubebysabr.hpp>
#include <ql/CashFlows/cmscoupon.hpp>
#include <ql/Indexes/euribor.hpp>
#include <ql/Optimization/method.hpp>
#include <ql/Optimization/problem.hpp>
#include <ql/Optimization/conjugategradient.hpp>
#include <ql/Optimization/simplex.hpp>
```

Include dependency graph for cmsmarket.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `std::vector< boost::shared_ptr< CashFlow > >` **Leg**



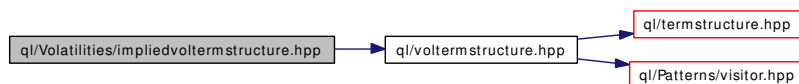
## 8.456 ql/Volatilities/impliedvoltermstructure.hpp File Reference

### 8.456.1 Detailed Description

Implied Black Vol Term Structure.

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

Include dependency graph for impliedvoltermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ImpliedVolTermStructure](#)  
*Implied vol term structure at a given date in the future.*

## 8.457 ql/Volatilities/localconstantvol.hpp File Reference

### 8.457.1 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**

### Classes

- class **LocalConstantVol**  
*Constant local volatility, no time-strike dependence.*

## 8.458 ql/Volatilities/localvolcurve.hpp File Reference

### 8.458.1 Detailed Description

Local volatility curve derived from a Black curve.

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for localvolcurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **LocalVolCurve**  
*Local volatility curve derived from a Black curve.*

## 8.459 ql/Volatilities/localvolsurface.hpp File Reference

### 8.459.1 Detailed Description

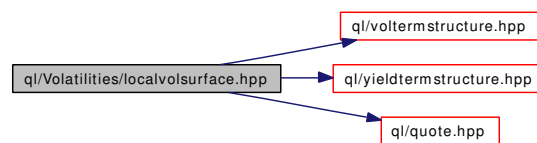
Local volatility surface derived from a Black vol surface.

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

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

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

Include dependency graph for localvolsurface.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LocalVolSurface](#)  
*Local volatility surface derived from a Black vol surface.*

## 8.460 ql/Volatilities/smilesection.hpp File Reference

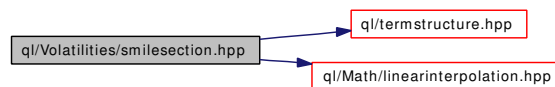
### 8.460.1 Detailed Description

Swaption volatility structure.

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

```
#include <ql/Math/linearinterpolation.hpp>
```

Include dependency graph for smilesection.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SmileSection](#)  
*swaption volatility smile section*

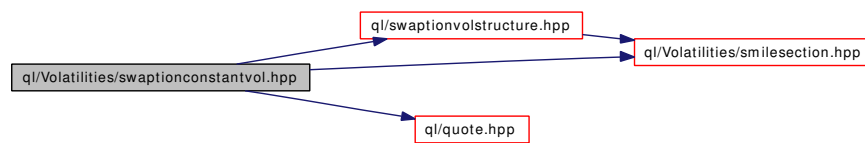
## 8.461 ql/Volatilities/swaptionconstantvol.hpp File Reference

### 8.461.1 Detailed Description

Constant swaption volatility.

```
#include <ql/swaptionvolstructure.hpp>
#include <ql/Volatilities/smilesection.hpp>
#include <ql/quote.hpp>
```

Include dependency graph for swaptionconstantvol.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwaptionConstantVolatility](#)  
*Constant swaption volatility, no time-strike dependence.*

## 8.462 ql/Volatilities/swaptionvolcube.hpp File Reference

### 8.462.1 Detailed Description

Swaption volatility cube.

```
#include <ql/Volatilities/swaptionvolmatrix.hpp>
```

```
#include <ql/Instruments/vanillaswap.hpp>
```

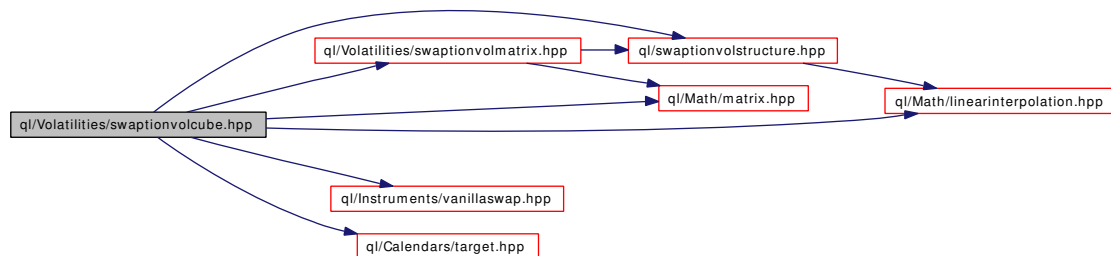
```
#include <ql/Math/linearinterpolation.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

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

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for swaptionvolcube.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwaptionVolatilityCube](#)

## 8.463 ql/Volatilities/swaptionvolcubebyabr.hpp File Reference

### 8.463.1 Detailed Description

Swaption volatility cube by Sabr.

```
#include <ql/Volatilities/swaptionvolmatrix.hpp>
```

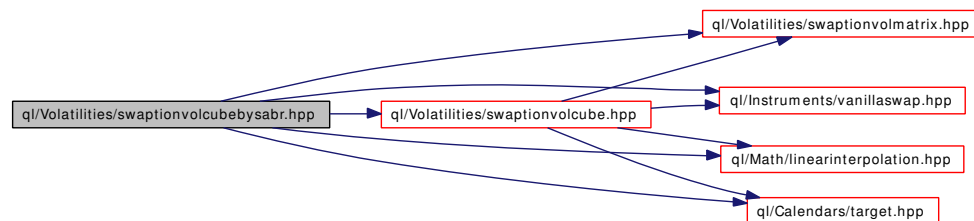
```
#include <ql/Instruments/vanillaswap.hpp>
```

```
#include <ql/Math/linearinterpolation.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/Volatilities/swaptionvolcube.hpp>
```

Include dependency graph for swaptionvolcubebyabr.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwaptionVolatilityCubeBySabr](#)



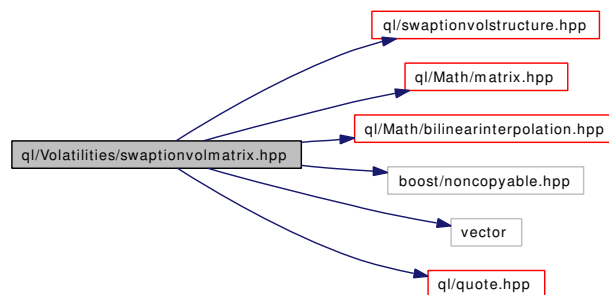
## 8.464 ql/Volatilities/swaptionvolmatrix.hpp File Reference

### 8.464.1 Detailed Description

Swaption at-the-money volatility matrix.

```
#include <ql/swaptionvolstructure.hpp>
#include <ql/Math/matrix.hpp>
#include <ql/Math/bilinearinterpolation.hpp>
#include <boost/noncopyable.hpp>
#include <vector>
#include <ql/quote.hpp>
```

Include dependency graph for swaptionvolmatrix.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwaptionVolatilityMatrix](#)  
*At-the-money swaption-volatility matrix.*

## 8.465 ql/volatilitymodel.hpp File Reference

### 8.465.1 Detailed Description

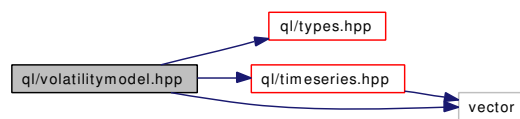
Volatility term structures.

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

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

```
#include <vector>
```

Include dependency graph for volatilitymodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LocalVolatilityEstimator](#)

## 8.466 ql/VolatilityModels/constantestimator.hpp File Reference

### 8.466.1 Detailed Description

Constant volatility estimator.

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

```
#include <vector>
```

Include dependency graph for constantestimator.hpp:



### Namespaces

- namespace `QuantLib`

### Classes

- class `ConstantEstimator`

## 8.467 ql/VolatilityModels/garch.hpp File Reference

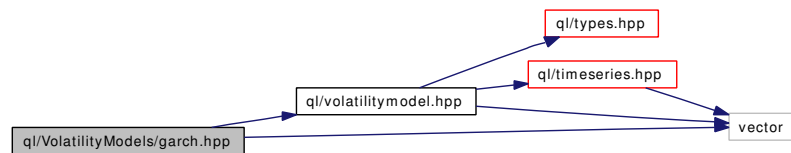
### 8.467.1 Detailed Description

GARCH volatility model.

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

```
#include <vector>
```

Include dependency graph for garch.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Garch11](#)  
*GARCH volatility model.*

## 8.468 ql/VolatilityModels/garmanklass.hpp File Reference

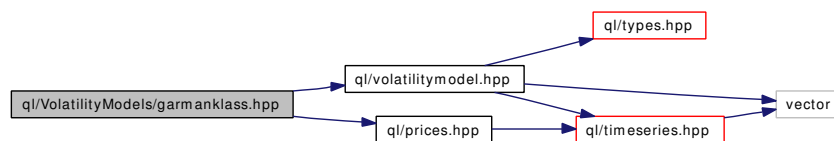
### 8.468.1 Detailed Description

Volatility estimators using high low data.

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

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

Include dependency graph for garmanklass.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GarmanKlassAbstract](#)
- class [GarmanKlassOpenClose](#)

## 8.469 ql/VolatilityModels/simplelocalestimator.hpp File Reference

### 8.469.1 Detailed Description

Constant volatility estimator.

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

```
#include <map>
```

Include dependency graph for simplelocalestimator.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SimpleLocalEstimator](#)

## 8.470 ql/voltermstructure.hpp File Reference

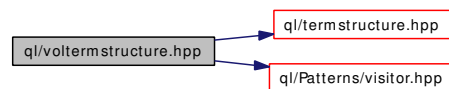
### 8.470.1 Detailed Description

Volatility term structures.

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

```
#include <ql/Patterns/visitor.hpp>
```

Include dependency graph for voltermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BlackVolTermStructure](#)  
*Black-volatility term structure.*
- class [BlackVolatilityTermStructure](#)  
*Black-volatility term structure.*
- class [BlackVarianceTermStructure](#)  
*Black variance term structure.*
- class [LocalVolTermStructure](#)  
*Local-volatility term structure.*

## 8.471 ql/yieldtermstructure.hpp File Reference

### 8.471.1 Detailed Description

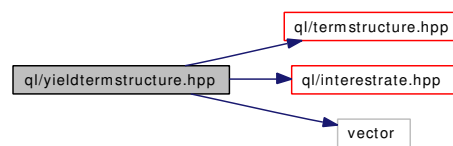
Interest-rate term structure.

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

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

```
#include <vector>
```

Include dependency graph for yieldtermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [YieldTermStructure](#)  
*Interest-rate term structure.*



## Chapter 9

# QuantLib Example Documentation

### 9.1 BermudanSwaption.cpp

This is an example of using the QuantLib short rate models.

```
1 /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
22 #define BOOST_LIB_DIAGNOSTIC
23 # include <ql/quantlib.hpp>
24 #undef BOOST_LIB_DIAGNOSTIC
25
26 #ifdef BOOST_MSVC
27 /* Uncomment the following lines to unmask floating-point
28    exceptions. Warning: unpredictable results can arise...
29
30    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
31    Is there anyone with a definitive word about this?
32 */
33 // #include <float.h>
34 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
35 #endif
36
37 #include <boost/timer.hpp>
38 #include <iostream>
39 #include <iomanip>
40
41 using namespace QuantLib;
42
43 #if defined(QL_ENABLE_SESSIONS)
44 namespace QuantLib {
45     Integer sessionId() { return 0; }
46 }
47 #endif
48
49 #endif
50
51
52 //Number of swaptions to be calibrated to...
53
54 Size numRows = 5;
55 Size numCols = 5;
56
57 Integer swapLengths[] = {
58     1,    2,    3,    4,    5};
59 Volatility swaptionVols[] = {
60     0.1490, 0.1340, 0.1228, 0.1189, 0.1148,
```

```

61 0.1290, 0.1201, 0.1146, 0.1108, 0.1040,
62 0.1149, 0.1112, 0.1070, 0.1010, 0.0957,
63 0.1047, 0.1021, 0.0980, 0.0951, 0.1270,
64 0.1000, 0.0950, 0.0900, 0.1230, 0.1160};
65
66 void calibrateModel(
67     const boost::shared_ptr<ShortRateModel>& model,
68     const std::vector<boost::shared_ptr<CalibrationHelper> >& helpers) {
69
70     LevenbergMarquardt om;
71     model->calibrate(helpers, om);
72
73     // Output the implied Black volatilities
74     for (Size i=0; i<numRows; i++) {
75         Size j = numCols - i - 1; // 1x5, 2x4, 3x3, 4x2, 5x1
76         Size k = i*numCols + j;
77         Real npv = helpers[i]->modelValue();
78         Volatility implied = helpers[i]->impliedVolatility(npv, 1e-4,
79                                                         1000, 0.05, 0.50);
80         Volatility diff = implied - swaptionVols[k];
81
82         std::cout << i+1 << "x" << swapLengths[j]
83                 << std::setprecision(5) << std::noshowpos
84                 << ": model " << std::setw(7) << io::volatility(implied)
85                 << ", market " << std::setw(7)
86                 << io::volatility(swaptionVols[k])
87                 << " (" << std::setw(7) << std::showpos
88                 << io::volatility(diff) << std::noshowpos << ")\n";
89     }
90 }
91
92 int main(int, char* [])
93 {
94     try {
95         QL_IO_INIT
96
97         boost::timer timer;
98         std::cout << std::endl;
99
100        Date todaysDate(15, February, 2002);
101        Calendar calendar = TARGET();
102        Date settlementDate(19, February, 2002);
103        Settings::instance().evaluationDate() = todaysDate;
104
105        // flat yield term structure impling 1x5 swap at 5%
106        boost::shared_ptr<Quote> flatRate(new SimpleQuote(0.04875825));
107        boost::shared_ptr<FlatForward> myTermStructure(
108            new FlatForward(settlementDate, Handle<Quote>(flatRate),
109                           Actual365Fixed()));
110        Handle<YieldTermStructure> rhTermStructure;
111        rhTermStructure.linkTo(myTermStructure);
112
113        // Define the ATM/OTM/ITM swaps
114        Frequency fixedLegFrequency = Annual;
115        BusinessDayConvention fixedLegConvention = Unadjusted;
116        BusinessDayConvention floatingLegConvention = ModifiedFollowing;
117        DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
118        Frequency floatingLegFrequency = Semiannual;
119        bool payFixedRate = true;
120        Rate dummyFixedRate = 0.03;
121        boost::shared_ptr<Xibor> indexSixMonths(
122            new Euribor6M(rhTermStructure));
123
124        Date startDate = calendar.advance(settlementDate, 1, Years,
125                                         floatingLegConvention);
126        Date maturity = calendar.advance(startDate, 5, Years,
127                                         floatingLegConvention);

```

```

128     Schedule fixedSchedule(startDate,maturity,Period(fixedLegFrequency),
129                             calendar,fixedLegConvention,fixedLegConvention,
130                             false,false);
131     Schedule floatSchedule(startDate,maturity,Period(floatingLegFrequency),
132                             calendar,floatingLegConvention,floatingLegConvention,
133                             false,false);
134
135     boost::shared_ptr<VanillaSwap> swap(new VanillaSwap(
136         payFixedRate, 1000.0,
137         fixedSchedule, dummyFixedRate, fixedLegDayCounter,
138         floatSchedule, indexSixMonths, 0.0,
139         indexSixMonths->dayCounter(), rhTermStructure));
140     Rate fixedATMRate = swap->fairRate();
141     Rate fixedOTMRate = fixedATMRate * 1.2;
142     Rate fixedITMRate = fixedATMRate * 0.8;
143
144     boost::shared_ptr<VanillaSwap> atmSwap(new VanillaSwap(
145         payFixedRate, 1000.0,
146         fixedSchedule, fixedATMRate, fixedLegDayCounter,
147         floatSchedule, indexSixMonths, 0.0,
148         indexSixMonths->dayCounter(), rhTermStructure));
149     boost::shared_ptr<VanillaSwap> otmSwap(new VanillaSwap(
150         payFixedRate, 1000.0,
151         fixedSchedule, fixedOTMRate, fixedLegDayCounter,
152         floatSchedule, indexSixMonths, 0.0,
153         indexSixMonths->dayCounter(), rhTermStructure));
154     boost::shared_ptr<VanillaSwap> itmSwap(new VanillaSwap(
155         payFixedRate, 1000.0,
156         fixedSchedule, fixedITMRate, fixedLegDayCounter,
157         floatSchedule, indexSixMonths, 0.0,
158         indexSixMonths->dayCounter(), rhTermStructure));
159
160     // defining the swaptions to be used in model calibration
161     std::vector<Period> swaptionMaturities;
162     swaptionMaturities.push_back(Period(1, Years));
163     swaptionMaturities.push_back(Period(2, Years));
164     swaptionMaturities.push_back(Period(3, Years));
165     swaptionMaturities.push_back(Period(4, Years));
166     swaptionMaturities.push_back(Period(5, Years));
167
168     std::vector<boost::shared_ptr<CalibrationHelper> > swaptions;
169
170     // List of times that have to be included in the timegrid
171     std::list<Time> times;
172
173     Size i;
174     for (i=0; i<numRows; i++) {
175         Size j = numCols - i -1; // 1x5, 2x4, 3x3, 4x2, 5x1
176         Size k = i*numCols + j;
177         boost::shared_ptr<Quote> vol(new SimpleQuote(swaptionVols[k]));
178         swaptions.push_back(boost::shared_ptr<CalibrationHelper>(new
179             SwaptionHelper(swaptionMaturities[i],
180                             Period(swapLenghts[j], Years),
181                             Handle<Quote>(vol),
182                             indexSixMonths,
183                             indexSixMonths->tenor(),
184                             indexSixMonths->dayCounter(),
185                             indexSixMonths->dayCounter(),
186                             rhTermStructure)));
187         swaptions.back()->addTimesTo(times);
188     }
189
190     // Building time-grid
191     TimeGrid grid(times.begin(), times.end(), 30);
192
193
194     // defining the models

```

```

195 boost::shared_ptr<G2> modelG2(new G2(rhTermStructure));
196 boost::shared_ptr<HullWhite> modelHW(new HullWhite(rhTermStructure));
197 boost::shared_ptr<HullWhite> modelHW2(new HullWhite(rhTermStructure));
198 boost::shared_ptr<BlackKarasinski> modelBK(
199     new BlackKarasinski(rhTermStructure));
200
201
202 // model calibrations
203
204 std::cout << "G2 (analytic formulae) calibration" << std::endl;
205 for (i=0; i<swaptions.size(); i++)
206     swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
207         new G2SwaptionEngine(modelG2, 6.0, 16)));
208
209 calibrateModel(modelG2, swaptions);
210 std::cout << "calibrated to:\n"
211     << "a = " << modelG2->params()[0] << ", "
212     << "sigma = " << modelG2->params()[1] << "\n"
213     << "b = " << modelG2->params()[2] << ", "
214     << "eta = " << modelG2->params()[3] << "\n"
215     << "rho = " << modelG2->params()[4]
216     << std::endl << std::endl;
217
218
219
220 std::cout << "Hull-White (analytic formulae) calibration" << std::endl;
221 for (i=0; i<swaptions.size(); i++)
222     swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
223         new JamshidianSwaptionEngine(modelHW)));
224
225 calibrateModel(modelHW, swaptions);
226 std::cout << "calibrated to:\n"
227     << "a = " << modelHW->params()[0] << ", "
228     << "sigma = " << modelHW->params()[1]
229     << std::endl << std::endl;
230
231 std::cout << "Hull-White (numerical) calibration" << std::endl;
232 for (i=0; i<swaptions.size(); i++)
233     swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
234         new TreeSwaptionEngine(modelHW2,grid)));
235
236 calibrateModel(modelHW2, swaptions);
237 std::cout << "calibrated to:\n"
238     << "a = " << modelHW2->params()[0] << ", "
239     << "sigma = " << modelHW2->params()[1]
240     << std::endl << std::endl;
241
242 std::cout << "Black-Karasinski (numerical) calibration" << std::endl;
243 for (i=0; i<swaptions.size(); i++)
244     swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
245         new TreeSwaptionEngine(modelBK,grid)));
246
247 calibrateModel(modelBK, swaptions);
248 std::cout << "calibrated to:\n"
249     << "a = " << modelBK->params()[0] << ", "
250     << "sigma = " << modelBK->params()[1]
251     << std::endl << std::endl;
252
253
254 // ATM Bermudan swaption pricing
255
256 std::cout << "Payer bermudan swaption "
257     << "struck at " << io::rate(fixedATMRate)
258     << " (ATM)" << std::endl;
259
260 std::vector<Date> bermudanDates;
261 const std::vector<boost::shared_ptr<CashFlow> >& leg =

```

```

262         swap->fixedLeg();
263     for (i=0; i<leg.size(); i++) {
264         boost::shared_ptr<Coupon> coupon =
265             boost::dynamic_pointer_cast<Coupon>(leg[i]);
266         bermudanDates.push_back(coupon->accrualStartDate());
267     }
268
269     boost::shared_ptr<Exercise> bermudanExercise(
270         new BermudanExercise(bermudanDates));
271
272     Swaption bermudanSwaption(atmSwap, bermudanExercise, rhTermStructure,
273         boost::shared_ptr<PricingEngine>());
274
275     // Do the pricing for each model
276
277     // G2 price the European swaption here, it should switch to bermudan
278     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
279         TreeSwaptionEngine(modelG2, 50)));
280     std::cout << "G2:      " << bermudanSwaption.NPV() << std::endl;
281
282     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
283         new TreeSwaptionEngine(modelHW, 50)));
284     std::cout << "HW:      " << bermudanSwaption.NPV() << std::endl;
285
286     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
287         TreeSwaptionEngine(modelHW2, 50)));
288     std::cout << "HW (num): " << bermudanSwaption.NPV() << std::endl;
289
290     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
291         TreeSwaptionEngine(modelBK, 50)));
292     std::cout << "BK:      " << bermudanSwaption.NPV() << std::endl;
293
294
295     // OTM Bermudan swaption pricing
296
297     std::cout << "Payer bermudan swaption "
298         << "struck at " << io::rate(fixedOTMRate)
299         << " (OTM)" << std::endl;
300
301     Swaption otmBermudanSwaption(otmSwap, bermudanExercise, rhTermStructure,
302         boost::shared_ptr<PricingEngine>());
303
304     // Do the pricing for each model
305     otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
306         new TreeSwaptionEngine(modelG2, 50)));
307     std::cout << "G2:      " << otmBermudanSwaption.NPV() << std::endl;
308
309     otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
310         new TreeSwaptionEngine(modelHW, 50)));
311     std::cout << "HW:      " << otmBermudanSwaption.NPV() << std::endl;
312
313     otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
314         new TreeSwaptionEngine(modelHW2, 50)));
315     std::cout << "HW (num): " << otmBermudanSwaption.NPV() << std::endl;
316
317     otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
318         new TreeSwaptionEngine(modelBK, 50)));
319     std::cout << "BK:      " << otmBermudanSwaption.NPV() << std::endl;
320
321
322     // ITM Bermudan swaption pricing
323
324     std::cout << "Payer bermudan swaption "
325         << "struck at " << io::rate(fixedITMRate)
326         << " (ITM)" << std::endl;
327
328     Swaption itmBermudanSwaption(itmSwap, bermudanExercise, rhTermStructure,

```

```

329                                     boost::shared_ptr<PricingEngine>());
330
331     // Do the pricing for each model
332     itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
333         new TreeSwaptionEngine(modelG2, 50)));
334     std::cout << "G2:      " << itmBermudanSwaption.NPV() << std::endl;
335
336     itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
337         new TreeSwaptionEngine(modelHW, 50)));
338     std::cout << "HW:      " << itmBermudanSwaption.NPV() << std::endl;
339
340     itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
341         new TreeSwaptionEngine(modelHW2, 50)));
342     std::cout << "HW (num): " << itmBermudanSwaption.NPV() << std::endl;
343
344     itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
345         new TreeSwaptionEngine(modelBK, 50)));
346     std::cout << "BK:      " << itmBermudanSwaption.NPV() << std::endl;
347
348     Real seconds = timer.elapsed();
349     Integer hours = int(seconds/3600);
350     seconds -= hours * 3600;
351     Integer minutes = int(seconds/60);
352     seconds -= minutes * 60;
353     std::cout << " \nRun completed in ";
354     if (hours > 0)
355         std::cout << hours << " h ";
356     if (hours > 0 || minutes > 0)
357         std::cout << minutes << " m ";
358     std::cout << std::fixed << std::setprecision(0)
359         << seconds << " s\n" << std::endl;
360
361     return 0;
362 } catch (std::exception& e) {
363     std::cout << e.what() << std::endl;
364     return 1;
365 } catch (...) {
366     std::cout << "unknown error" << std::endl;
367     return 1;
368 }
369 }
370

```

## 9.2 ConvertibleBonds.cpp

This example evaluates convertible bond prices.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
21 // the only header you need to use QuantLib
22 #define BOOST_LIB_DIAGNOSTIC
23 # include <ql/quantlib.hpp>
24 #undef BOOST_LIB_DIAGNOSTIC
25
26 #ifdef BOOST_MSVC
27 /* Uncomment the following lines to unmask floating-point
28    exceptions. Warning: unpredictable results can arise...
29
30    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
31    Is there anyone with a definitive word about this?
32 */
33 // #include <float.h>
34 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
35 #endif
36
37 #include <boost/timer.hpp>
38 #include <iostream>
39 #include <iomanip>
40
41 #define LENGTH(a) (sizeof(a)/sizeof(a[0]))
42
43 using namespace QuantLib;
44
45 #if defined(QL_ENABLE_SESSIONS)
46 namespace QuantLib {
47
48     Integer sessionId() { return 0; }
49
50 }
51 #endif
52
53
54 int main()
55 {
56     try {
57
58         QL_IO_INIT
59
60         boost::timer timer;
61         std::cout << std::endl;
62
63         Option::Type type(Option::Put);
64         Real underlying = 36.0;
65         Real spreadRate = 0.005;
66
67         Spread dividendYield = 0.02;
68         Rate riskFreeRate = 0.06;
69         Volatility volatility = 0.20;
70
71         Integer settlementDays = 3;
72         Integer length = 5;
73         Real redemption = 100.0;
74         Real conversionRatio = redemption/underlying; // at the money
75
76         // set up dates/schedules
77         Calendar calendar = TARGET();
78         Date today = calendar.adjust(Date::todaysDate());
79
80         Settings::instance().evaluationDate() = today;

```

```

81     Date settlementDate = calendar.advance(today, settlementDays, Days);
82     Date exerciseDate = calendar.advance(settlementDate, length, Years);
83     Date issueDate = calendar.advance(exerciseDate, -length, Years);
84
85     BusinessDayConvention convention = ModifiedFollowing;
86
87     Frequency frequency = Annual;
88
89     Schedule schedule(issueDate, exerciseDate, Period(frequency), calendar,
90                       convention, convention, true, false);
91
92     DividendSchedule dividends;
93     CallabilitySchedule callability;
94
95     std::vector<Real> coupons(1, 0.05);
96
97     DayCounter bondDayCount = Thirty360();
98
99     Integer callLength[] = { 2, 4 }; // Call dates, years 2, 4.
100    Integer putLength[] = { 3 }; // Put dates year 3
101
102    Real callPrices[] = { 101.5, 100.85 };
103    Real putPrices[] = { 105.0 };
104
105    // Load call schedules
106    for (Size i=0; i<LENGTH(callLength); i++) {
107        callability.push_back(
108            boost::shared_ptr<Callability>(
109                new SoftCallability(Callability::Price(
110                    callPrices[i],
111                    Callability::Price::Clean),
112                    schedule.date(callLength[i]),
113                    1.20)));
114    }
115
116    for (Size j=0; j<LENGTH(putLength); j++) {
117        callability.push_back(
118            boost::shared_ptr<Callability>(
119                new Callability(Callability::Price(
120                    putPrices[j],
121                    Callability::Price::Clean),
122                    Callability::Put,
123                    schedule.date(putLength[j]))));
124    }
125
126    // Assume dividends are paid every 6 months.
127    for (Date d = today + 6*Months; d < exerciseDate; d += 6*Months) {
128        dividends.push_back(
129            boost::shared_ptr<Dividend>(new FixedDividend(1.0, d)));
130    }
131
132    DayCounter dayCounter = Actual365Fixed();
133    Time maturity = dayCounter.yearFraction(settlementDate,
134                                           exerciseDate);
135
136    std::cout << "option type = " << type << std::endl;
137    std::cout << "Time to maturity = " << maturity
138              << std::endl;
139    std::cout << "Underlying price = " << underlying
140              << std::endl;
141    std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
142              << std::endl;
143    std::cout << "Dividend yield = " << io::rate(dividendYield)
144              << std::endl;
145    std::cout << "Volatility = " << io::volatility(volatility)
146              << std::endl;
147    std::cout << std::endl;

```



```

148
149     std::string method;
150     std::cout << std::endl ;
151
152     // write column headings
153     Size widths[] = { 35, 14, 14 };
154     Size totalWidth = widths[0] + widths[1] + widths[2];
155     std::string rule(totalWidth, '-'), dblrule(totalWidth, '=');
156
157     std::cout << dblrule << std::endl;
158     std::cout << "Tsiveriotis-Fernandes method" << std::endl;
159     std::cout << dblrule << std::endl;
160     std::cout << std::setw(widths[0]) << std::left << "Tree type"
161               << std::setw(widths[1]) << std::left << "European"
162               << std::setw(widths[1]) << std::left << "American"
163               << std::endl;
164     std::cout << rule << std::endl;
165
166     boost::shared_ptr<Exercise> exercise(
167         new EuropeanExercise(exerciseDate));
168     boost::shared_ptr<Exercise> amExercise(
169         new AmericanExercise(settlementDate,
170                             exerciseDate));
171
172     Handle<Quote> underlyingH(
173         boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
174
175     Handle<YieldTermStructure> flatTermStructure(
176         boost::shared_ptr<YieldTermStructure>(
177             new FlatForward(settlementDate, riskFreeRate, dayCounter)));
178
179     Handle<YieldTermStructure> flatDividendTS(
180         boost::shared_ptr<YieldTermStructure>(
181             new FlatForward(settlementDate, dividendYield, dayCounter)));
182
183     Handle<BlackVolTermStructure> flatVolTS(
184         boost::shared_ptr<BlackVolTermStructure>(
185             new BlackConstantVol(settlementDate, volatility, dayCounter)));
186
187
188     boost::shared_ptr<StochasticProcess> stochasticProcess(
189         new BlackScholesMertonProcess(underlyingH,
190                                       flatDividendTS,
191                                       flatTermStructure,
192                                       flatVolTS));
193
194     Size timeSteps = 801;
195
196     Handle<Quote> creditSpread(
197         boost::shared_ptr<Quote>(new SimpleQuote(spreadRate)));
198
199     boost::shared_ptr<Quote> rate(new SimpleQuote(riskFreeRate));
200
201     Handle<YieldTermStructure> discountCurve(
202         boost::shared_ptr<YieldTermStructure>(
203             new FlatForward(today, Handle<Quote>(rate), dayCounter)));
204
205     boost::shared_ptr<PricingEngine> engine(
206         new BinomialConvertibleEngine<JarrowRudd>(timeSteps));
207
208     ConvertibleFixedCouponBond europeanBond(
209         stochasticProcess, exercise, engine,
210         conversionRatio, dividends, callability,
211         creditSpread, issueDate, settlementDays,
212         coupons, bondDayCount, schedule, redemption);
213
214     ConvertibleFixedCouponBond americanBond(

```

```

215         stochasticProcess, amExercise, engine,
216         conversionRatio, dividends, callability,
217         creditSpread, issueDate, settlementDays,
218         coupons, bondDayCount, schedule, redemption);
219
220     method = "Jarrow-Rudd";
221     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
222         new BinomialConvertibleEngine<JarrowRudd>(timeSteps)));
223     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
224         new BinomialConvertibleEngine<JarrowRudd>(timeSteps)));
225     std::cout << std::setw(widths[0]) << std::left << method
226               << std::fixed
227               << std::setw(widths[1]) << std::left << europeanBond.NPV()
228               << std::setw(widths[2]) << std::left << americanBond.NPV()
229               << std::endl;
230
231     method = "Cox-Ross-Rubinstein";
232     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
233         new BinomialConvertibleEngine<CoxRossRubinstein>(timeSteps)));
234     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
235         new BinomialConvertibleEngine<CoxRossRubinstein>(timeSteps)));
236     std::cout << std::setw(widths[0]) << std::left << method
237               << std::fixed
238               << std::setw(widths[1]) << std::left << europeanBond.NPV()
239               << std::setw(widths[2]) << std::left << americanBond.NPV()
240               << std::endl;
241
242     method = "Additive equiprobabilities";
243     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
244         new BinomialConvertibleEngine<AdditiveEQPBinoialTree>(timeSteps)));
245     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
246         new BinomialConvertibleEngine<AdditiveEQPBinoialTree>(timeSteps)));
247     std::cout << std::setw(widths[0]) << std::left << method
248               << std::fixed
249               << std::setw(widths[1]) << std::left << europeanBond.NPV()
250               << std::setw(widths[2]) << std::left << americanBond.NPV()
251               << std::endl;
252
253     method = "Trigeorgis";
254     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
255         new BinomialConvertibleEngine<Trigeorgis>(timeSteps)));
256     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
257         new BinomialConvertibleEngine<Trigeorgis>(timeSteps)));
258     std::cout << std::setw(widths[0]) << std::left << method
259               << std::fixed
260               << std::setw(widths[1]) << std::left << europeanBond.NPV()
261               << std::setw(widths[2]) << std::left << americanBond.NPV()
262               << std::endl;
263
264     method = "Tian";
265     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
266         new BinomialConvertibleEngine<Tian>(timeSteps)));
267     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
268         new BinomialConvertibleEngine<Tian>(timeSteps)));
269     std::cout << std::setw(widths[0]) << std::left << method
270               << std::fixed
271               << std::setw(widths[1]) << std::left << europeanBond.NPV()
272               << std::setw(widths[2]) << std::left << americanBond.NPV()
273               << std::endl;
274
275     method = "Leisen-Reimer";
276     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
277         new BinomialConvertibleEngine<LeisenReimer>(timeSteps)));
278     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
279         new BinomialConvertibleEngine<LeisenReimer>(timeSteps)));
280     std::cout << std::setw(widths[0]) << std::left << method
281               << std::fixed

```

```
282         << std::setw(widths[1]) << std::left << europeanBond.NPV()
283         << std::setw(widths[2]) << std::left << americanBond.NPV()
284         << std::endl;
285
286     std::cout << dblrule << std::endl;
287
288     Real seconds = timer.elapsed();
289     Integer hours = int(seconds/3600);
290     seconds -= hours * 3600;
291     Integer minutes = int(seconds/60);
292     seconds -= minutes * 60;
293     std::cout << " \nRun completed in ";
294     if (hours > 0)
295         std::cout << hours << " h ";
296     if (hours > 0 || minutes > 0)
297         std::cout << minutes << " m ";
298     std::cout << std::fixed << std::setprecision(0)
299         << seconds << " s\n" << std::endl;
300
301     return 0;
302 } catch (std::exception& e) {
303     std::cout << e.what() << std::endl;
304     return 1;
305 } catch (...) {
306     std::cout << "unknown error" << std::endl;
307     return 1;
308 }
309
310 }
311
```

### 9.3 DiscreteHedging.cpp

This is an example of using the QuantLib Monte Carlo framework.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
21 /* This example computes profit and loss of a discrete interval hedging
22    strategy and compares with the results of Derman & Kamal's (Goldman Sachs
23    Equity Derivatives Research) Research Note: "When You Cannot Hedge
24    Continuously: The Corrections to Black-Scholes"
25    http://www.ederman.com/emanuelderman/GSQSpapers/when_you_cannot_hedge.pdf
26
27    Suppose an option hedger sells an European option and receives the
28    Black-Scholes value as the options premium.
29    Then he follows a Black-Scholes hedging strategy, rehedging at discrete,
30    evenly spaced time intervals as the underlying stock changes. At
31    expiration, the hedger delivers the option payoff to the option holder,
32    and unwinds the hedge. We are interested in understanding the final
33    profit or loss of this strategy.
34
35    If the hedger had followed the exact Black-Scholes replication strategy,
36    re-hedging continuously as the underlying stock evolved towards its final
37    value at expiration, then, no matter what path the stock took, the final
38    P&L would be exactly zero. When the replication strategy deviates from
39    the exact Black-Scholes method, the final P&L may deviate from zero. This
40    deviation is called the replication error. When the hedger rebalances at
41    discrete rather than continuous intervals, the hedge is imperfect and the
42    replication is inexact. The more often hedging occurs, the smaller the
43    replication error.
44
45    We examine the range of possibilities, computing the replication error.
46 */
47
48 // the only header you need to use QuantLib
49 #define BOOST_LIB_DIAGNOSTIC
50 # include <ql/quantlib.hpp>
51 #undef BOOST_LIB_DIAGNOSTIC
52
53 #ifdef BOOST_MSVC
54 /* Uncomment the following lines to unmask floating-point
55    exceptions. Warning: unpredictable results can arise...
56
57    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
58    Is there anyone with a definitive word about this?
59 */
60 // #include <float.h>
61 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
62 #endif
63
64 #include <boost/timer.hpp>
65 #include <iostream>
66 #include <iomanip>
67
68 using namespace QuantLib;
69
70 #if defined(QL_ENABLE_SESSIONS)
71 namespace QuantLib {
72     Integer sessionId() { return 0; }
73 }
74 #endif
75
76 #endif
77
78
79 /* The ReplicationError class carries out Monte Carlo simulations to evaluate
80    the outcome (the replication error) of the discrete hedging strategy over

```

```

81 different, randomly generated scenarios of future stock price evolution.
82 */
83 class ReplicationError
84 {
85 public:
86     ReplicationError(Option::Type type,
87                     Time maturity,
88                     Real strike,
89                     Real s0,
90                     Volatility sigma,
91                     Rate r)
92     : maturity_(maturity), payoff_(type, strike), s0_(s0),
93       sigma_(sigma), r_(r) {
94
95         // value of the option
96         DiscountFactor rDiscount = std::exp(-r_*maturity_);
97         DiscountFactor qDiscount = 1.0;
98         Real forward = s0_*qDiscount/rDiscount;
99         Real variance = sigma_*sigma_*maturity_;
100         boost::shared_ptr<StrikedTypePayoff> payoff(
101             new PlainVanillaPayoff(payoff_));
102         BlackFormula black(forward,rDiscount,variance,payoff);
103         std::cout << "Option value: " << black.value() << std::endl;
104
105         // store option's vega, since Derman and Kamal's formula needs it
106         vega_ = black.vega(maturity_);
107
108         std::cout << std::endl;
109         std::cout <<
110             "          |          | P&L \t| P&L          | Derman&Kamal | P&L"
111             "          \t| P&L" << std::endl;
112
113         std::cout <<
114             "samples | trades | Mean \t| Std Dev | Formula          |"
115             " skewness \t| kurt." << std::endl;
116
117         std::cout << "-----"
118             "-----" << std::endl;
119     }
120
121     // the actual replication error computation
122     void compute(Size nTimeSteps, Size nSamples);
123 private:
124     Time maturity_;
125     PlainVanillaPayoff payoff_;
126     Real s0_;
127     Volatility sigma_;
128     Rate r_;
129     Real vega_;
130 };
131
132 // The key for the MonteCarlo simulation is to have a PathPricer that
133 // implements a value(const Path& path) method.
134 // This method prices the portfolio for each Path of the random variable
135 class ReplicationPathPricer : public PathPricer<Path> {
136 public:
137     // real constructor
138     ReplicationPathPricer(Option::Type type,
139                         Real strike,
140                         Rate r,
141                         Time maturity,
142                         Volatility sigma)
143     : type_(type), strike_(strike),
144       r_(r), maturity_(maturity), sigma_(sigma) {
145         QL_REQUIRE(strike_ > 0.0, "strike must be positive");
146         QL_REQUIRE(r_ >= 0.0,
147             "risk free rate (r) must be positive or zero");

```

```

148     QL_REQUIRE(maturity_ > 0.0, "maturity must be positive");
149     QL_REQUIRE(sigma_ >= 0.0,
150               "volatility (sigma) must be positive or zero");
151
152 }
153 // The value() method encapsulates the pricing code
154 Real operator()(const Path& path) const;
155
156 private:
157     Option::Type type_;
158     Real strike_;
159     Rate r_;
160     Time maturity_;
161     Volatility sigma_;
162 };
163
164
165 // Compute Replication Error as in the Derman and Kamal's research note
166 int main(int, char* [])
167 {
168     try {
169         QL_IO_INIT
170
171         boost::timer timer;
172         std::cout << std::endl;
173
174         Time maturity = 1.0/12.0;    // 1 month
175         Real strike = 100;
176         Real underlying = 100;
177         Volatility volatility = 0.20; // 20%
178         Rate riskFreeRate = 0.05;    // 5%
179         ReplicationError rp(Option::Call, maturity, strike, underlying,
180                           volatility, riskFreeRate);
181
182         Size scenarios = 50000;
183         Size hedgesNum;
184
185         hedgesNum = 21;
186         rp.compute(hedgesNum, scenarios);
187
188         hedgesNum = 84;
189         rp.compute(hedgesNum, scenarios);
190
191         Real seconds = timer.elapsed();
192         Integer hours = int(seconds/3600);
193         seconds -= hours * 3600;
194         Integer minutes = int(seconds/60);
195         seconds -= minutes * 60;
196         std::cout << " \nRun completed in ";
197         if (hours > 0)
198             std::cout << hours << " h ";
199         if (hours > 0 || minutes > 0)
200             std::cout << minutes << " m ";
201         std::cout << std::fixed << std::setprecision(0)
202             << seconds << " s\n" << std::endl;
203
204         return 0;
205     } catch (std::exception& e) {
206         std::cout << e.what() << std::endl;
207         return 1;
208     } catch (...) {
209         std::cout << "unknown error" << std::endl;
210         return 1;
211     }
212 }
213
214

```

```

215 /* The actual computation of the Profit&Loss for each single path.
216
217     In each scenario N reheding trades spaced evenly in time over
218     the life of the option are carried out, using the Black-Scholes
219     hedge ratio.
220 */
221 Real ReplicationPathPricer::operator()(const Path& path) const {
222
223     Size n = path.length()-1;
224     QL_REQUIRE(n>0, "the path cannot be empty");
225
226     // discrete hedging interval
227     Time dt = maturity_/n;
228
229     // For simplicity, we assume the stock pays no dividends.
230     Rate stockDividendYield = 0.0;
231
232     // let's start
233     Time t = 0;
234
235     // stock value at t=0
236     Real stock = path.front();
237
238     // money account at t=0
239     Real money_account = 0.0;
240
241     /***** the initial deal *****/
242     // option fair price (Black-Scholes) at t=0
243     DiscountFactor rDiscount = std::exp(-r_*maturity_);
244     DiscountFactor qDiscount = std::exp(-stockDividendYield*maturity_);
245     Real forward = stock*qDiscount/rDiscount;
246     Real variance = sigma_*sigma_*maturity_;
247     boost::shared_ptr<StrikedTypePayoff> payoff(
248         new PlainVanillaPayoff(type_,strike_));
249     BlackFormula black(forward,rDiscount,variance,payoff);
250     // sell the option, cash in its premium
251     money_account += black.value();
252     // compute delta
253     Real delta = black.delta(stock);
254     // delta-hedge the option buying stock
255     Real stockAmount = delta;
256     money_account -= stockAmount*stock;
257
258     /***** hedging during option life *****/
259     for (Size step = 0; step < n-1; step++){
260
261         // time flows
262         t += dt;
263
264         // accruing on the money account
265         money_account *= std::exp( r_*dt );
266
267         // stock growth:
268         stock = path[step+1];
269
270         // recalculate option value at the current stock value,
271         // and the current time to maturity
272         rDiscount = std::exp(-r_*(maturity_-t));
273         qDiscount = std::exp(-stockDividendYield*(maturity_-t));
274         forward = stock*qDiscount/rDiscount;
275         variance = sigma_*sigma_*(maturity_-t);
276         BlackFormula black(forward,rDiscount,variance,payoff);
277

```

```

282         // recalculate delta
283         delta = black.delta(stock);
284
285         // re-hedging
286         money_account -= (delta - stockAmount)*stock;
287         stockAmount = delta;
288     }
289
290     /*****
291     /** option expiration **/
292     *****/
293     // last accrual on my money account
294     money_account *= std::exp( r_*dt );
295     // last stock growth
296     stock = path[n];
297
298     // the hedger delivers the option payoff to the option holder
299     Real optionPayoff = PlainVanillaPayoff(type_, strike_)(stock);
300     money_account -= optionPayoff;
301
302     // and unwinds the hedge selling his stock position
303     money_account += stockAmount*stock;
304
305     // final Profit&Loss
306     return money_account;
307 }
308
309
310 // The computation over nSamples paths of the P&L distribution
311 void ReplicationError::compute(Size nTimeSteps, Size nSamples)
312 {
313     QL_REQUIRE(nTimeSteps>0, "the number of steps must be > 0");
314
315     // hedging interval
316     // Time tau = maturity_ / nTimeSteps;
317
318     /* Black-Scholes framework: the underlying stock price evolves
319        lognormally with a fixed known volatility that stays constant
320        throughout time.
321     */
322     Date today = Date::todaysDate();
323     DayCounter dayCount = Actual365Fixed();
324     Handle<Quote> stateVariable(
325         boost::shared_ptr<Quote>(new SimpleQuote(s0_)));
326     Handle<YieldTermStructure> riskFreeRate(
327         boost::shared_ptr<YieldTermStructure>(
328             new FlatForward(today, r_, dayCount)));
329     Handle<YieldTermStructure> dividendYield(
330         boost::shared_ptr<YieldTermStructure>(
331             new FlatForward(today, 0.0, dayCount)));
332     Handle<BlackVolTermStructure> volatility(
333         boost::shared_ptr<BlackVolTermStructure>(
334             new BlackConstantVol(today, sigma_, dayCount)));
335     boost::shared_ptr<StochasticProcess1D> diffusion(
336         new BlackScholesMertonProcess(stateVariable, dividendYield,
337             riskFreeRate, volatility));
338
339     // Black Scholes equation rules the path generator:
340     // at each step the log of the stock
341     // will have drift and sigma^2 variance
342     PseudoRandom::rsg_type rsg =
343         PseudoRandom::make_sequence_generator(nTimeSteps, 0);
344
345     bool brownianBridge = false;
346
347     typedef SingleVariate<PseudoRandom>::path_generator_type generator_type;
348     boost::shared_ptr<generator_type> myPathGenerator(new

```



```

349         generator_type(diffusion, maturity_, nTimeSteps,
350                         rsg, brownianBridge));
351
352     // The replication strategy's Profit&Loss is computed for each path
353     // of the stock. The path pricer knows how to price a path using its
354     // value() method
355     boost::shared_ptr<PathPricer<Path> > myPathPricer(new
356         ReplicationPathPricer(payoff_.optionType(), payoff_.strike(),
357                               r_, maturity_, sigma_));
358
359     // a statistics accumulator for the path-dependant Profit&Loss values
360     Statistics statisticsAccumulator;
361
362     // The OneFactorMontecarloModel generates paths using myPathGenerator
363     // each path is priced using myPathPricer
364     // prices will be accumulated into statisticsAccumulator
365     OneFactorMonteCarloOption MCSimulation(myPathGenerator,
366                                           myPathPricer,
367                                           statisticsAccumulator,
368                                           false);
369
370     // the model simulates nSamples paths
371     MCSimulation.addSamples(nSamples);
372
373     // the sampleAccumulator method of OneFactorMonteCarloOption_old
374     // gives access to all the methods of statisticsAccumulator
375     Real PLMean = MCSimulation.sampleAccumulator().mean();
376     Real PLStDev = MCSimulation.sampleAccumulator().standardDeviation();
377     Real PLSkew = MCSimulation.sampleAccumulator().skewness();
378     Real PLKurt = MCSimulation.sampleAccumulator().kurtosis();
379
380     // Derman and Kamal's formula
381     Real theorStd = std::sqrt(M_PI/4/nTimeSteps)*vega_*sigma_;
382
383
384     std::cout << std::fixed
385               << nSamples << "\t| "
386               << nTimeSteps << "\t | "
387               << std::setprecision(3) << PLMean << " \t| "
388               << std::setprecision(2) << PLStDev << " \t | "
389               << std::setprecision(2) << theorStd << " \t | "
390               << std::setprecision(2) << PLSkew << " \t| "
391               << std::setprecision(2) << PLKurt << std::endl;
392 }

```

## 9.4 EquityOption.cpp

This example evaluates European, American and Bermudan options using different methods

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 // the only header you need to use QuantLib
21 #define BOOST_LIB_DIAGNOSTIC
22 # include <ql/quantlib.hpp>
23 #undef BOOST_LIB_DIAGNOSTIC
24
25 #ifdef BOOST_MSVC
26 /* Uncomment the following lines to unmask floating-point
27    exceptions. Warning: unpredictable results can arise...
28
29    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
30    Is there anyone with a definitive word about this?
31 */
32 // #include <float.h>
33 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
34 #endif
35
36 #include <boost/timer.hpp>
37 #include <iostream>
38 #include <iomanip>
39
40 using namespace QuantLib;
41
42 #if defined(QL_ENABLE_SESSIONS)
43 namespace QuantLib {
44
45     Integer sessionId() { return 0; }
46
47 }
48 #endif
49
50
51 int main(int, char* [])
52 {
53     try {
54         QL_IO_INIT
55
56         boost::timer timer;
57         std::cout << std::endl;
58
59         // our options
60         Option::Type type(Option::Put);
61         Real underlying = 36;
62         Real strike = 40;
63         Spread dividendYield = 0.00;
64         Rate riskFreeRate = 0.06;
65         Volatility volatility = 0.20;
66
67         Date todaysDate(15, May, 1998);
68         Date settlementDate(17, May, 1998);
69         Settings::instance().evaluationDate() = todaysDate;
70
71         Date maturity(17, May, 1999);
72         DayCounter dayCounter = Actual365Fixed();
73
74         std::cout << "Option type = " << type << std::endl;
75         std::cout << "Maturity = " << maturity << std::endl;
76         std::cout << "Underlying price = " << underlying << std::endl;
77         std::cout << "Strike = " << strike << std::endl;
78         std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
79             << std::endl;

```

```

80     std::cout << "Dividend yield = " << io::rate(dividendYield)
81         << std::endl;
82     std::cout << "Volatility = " << io::volatility(volatility)
83         << std::endl;
84     std::cout << std::endl;
85
86     std::string method;
87
88     std::cout << std::endl ;
89
90     // write column headings
91     Size widths[] = { 35, 14, 14, 14 };
92     std::cout << std::setw(widths[0]) << std::left << "Method"
93         << std::setw(widths[1]) << std::left << "European"
94         << std::setw(widths[2]) << std::left << "Bermudan"
95         << std::setw(widths[3]) << std::left << "American"
96         << std::endl;
97
98     std::vector<Date> exerciseDates;
99     for (Integer i=1; i<=4; i++)
100         exerciseDates.push_back(settlementDate + 3*i*Months);
101
102     boost::shared_ptr<Exercise> europeanExercise(
103         new EuropeanExercise(maturity));
104
105     boost::shared_ptr<Exercise> bermudanExercise(
106         new BermudanExercise(exerciseDates));
107
108     boost::shared_ptr<Exercise> americanExercise(
109         new AmericanExercise(settlementDate,
110             maturity));
111
112     Handle<Quote> underlyingH(
113         boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
114
115     // bootstrap the yield/dividend/vol curves
116     Handle<YieldTermStructure> flatTermStructure(
117         boost::shared_ptr<YieldTermStructure>(
118             new FlatForward(settlementDate, riskFreeRate, dayCounter)));
119     Handle<YieldTermStructure> flatDividendTS(
120         boost::shared_ptr<YieldTermStructure>(
121             new FlatForward(settlementDate, dividendYield, dayCounter)));
122     Handle<BlackVolTermStructure> flatVolTS(
123         boost::shared_ptr<BlackVolTermStructure>(
124             new BlackConstantVol(settlementDate, volatility, dayCounter)));
125
126     boost::shared_ptr<StrikedTypePayoff> payoff(
127         new PlainVanillaPayoff(type, strike));
128
129     boost::shared_ptr<StochasticProcess> stochasticProcess(
130         new BlackScholesMertonProcess(underlyingH, flatDividendTS,
131             flatTermStructure, flatVolTS));
132
133     // options
134
135     VanillaOption europeanOption(stochasticProcess, payoff,
136         europeanExercise);
137
138     VanillaOption bermudanOption(stochasticProcess, payoff,
139         bermudanExercise);
140
141     VanillaOption americanOption(stochasticProcess, payoff,
142         americanExercise);
143
144     // Analytic formulas:
145
146     // Black-Scholes for European

```

```

147     method = "Black-Scholes";
148     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
149         new AnalyticEuropeanEngine));
150     std::cout << std::setw(widths[0]) << std::left << method
151         << std::fixed
152         << std::setw(widths[1]) << std::left << europeanOption.NPV()
153         << std::setw(widths[2]) << std::left << "N/A"
154         << std::setw(widths[3]) << std::left << "N/A"
155         << std::endl;
156
157     // Barone-Adesi and Whaley approximation for American
158     method = "Barone-Adesi/Whaley";
159     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
160         new BaroneAdesiWhaleyApproximationEngine));
161     std::cout << std::setw(widths[0]) << std::left << method
162         << std::fixed
163         << std::setw(widths[1]) << std::left << "N/A"
164         << std::setw(widths[2]) << std::left << "N/A"
165         << std::setw(widths[3]) << std::left << americanOption.NPV()
166         << std::endl;
167
168     // Bjerksund and Stensland approximation for American
169     method = "Bjerksund/Stensland";
170     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
171         new BjerksundStenslandApproximationEngine));
172     std::cout << std::setw(widths[0]) << std::left << method
173         << std::fixed
174         << std::setw(widths[1]) << std::left << "N/A"
175         << std::setw(widths[2]) << std::left << "N/A"
176         << std::setw(widths[3]) << std::left << americanOption.NPV()
177         << std::endl;
178
179     // Integral
180
181     method = "Integral";
182     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
183         new IntegralEngine));
184     std::cout << std::setw(widths[0]) << std::left << method
185         << std::fixed
186         << std::setw(widths[1]) << std::left << europeanOption.NPV()
187         << std::setw(widths[2]) << std::left << "N/A"
188         << std::setw(widths[3]) << std::left << "N/A"
189         << std::endl;
190
191     // Finite differences
192
193     Size timeSteps = 801;
194
195     method = "Finite differences";
196     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
197         new FDEuropeanEngine(timeSteps,timeSteps-1)));
198     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
199         new FDBermudanEngine(timeSteps,timeSteps-1)));
200     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
201         new FDAmericanEngine(timeSteps,timeSteps-1)));
202     std::cout << std::setw(widths[0]) << std::left << method
203         << std::fixed
204         << std::setw(widths[1]) << std::left << europeanOption.NPV()
205         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
206         << std::setw(widths[3]) << std::left << americanOption.NPV()
207         << std::endl;
208
209     // Binomial method
210
211     method = "Binomial Jarrow-Rudd";
212     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
213         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));

```

```

214     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
215         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
216     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
217         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
218     std::cout << std::setw(widths[0]) << std::left << method
219         << std::fixed
220         << std::setw(widths[1]) << std::left << europeanOption.NPV()
221         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
222         << std::setw(widths[3]) << std::left << americanOption.NPV()
223         << std::endl;
224
225     method = "Binomial Cox-Ross-Rubinstein";
226     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
227         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
228     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
229         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
230     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
231         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
232     std::cout << std::setw(widths[0]) << std::left << method
233         << std::fixed
234         << std::setw(widths[1]) << std::left << europeanOption.NPV()
235         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
236         << std::setw(widths[3]) << std::left << americanOption.NPV()
237         << std::endl;
238
239     method = "Additive equiprobabilities";
240     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
241         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
242     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
243         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
244     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
245         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
246     std::cout << std::setw(widths[0]) << std::left << method
247         << std::fixed
248         << std::setw(widths[1]) << std::left << europeanOption.NPV()
249         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
250         << std::setw(widths[3]) << std::left << americanOption.NPV()
251         << std::endl;
252
253     method = "Binomial Trigeorgis";
254     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
255         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
256     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
257         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
258     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
259         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
260     std::cout << std::setw(widths[0]) << std::left << method
261         << std::fixed
262         << std::setw(widths[1]) << std::left << europeanOption.NPV()
263         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
264         << std::setw(widths[3]) << std::left << americanOption.NPV()
265         << std::endl;
266
267     method = "Binomial Tian";
268     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
269         new BinomialVanillaEngine<Tian>(timeSteps)));
270     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
271         new BinomialVanillaEngine<Tian>(timeSteps)));
272     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
273         new BinomialVanillaEngine<Tian>(timeSteps)));
274     std::cout << std::setw(widths[0]) << std::left << method
275         << std::fixed
276         << std::setw(widths[1]) << std::left << europeanOption.NPV()
277         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
278         << std::setw(widths[3]) << std::left << americanOption.NPV()
279         << std::endl;
280

```

```

281     method = "Binomial Leisen-Reimer";
282     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
283         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
284     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
285         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
286     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
287         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
288     std::cout << std::setw(widths[0]) << std::left << method
289         << std::fixed
290         << std::setw(widths[1]) << std::left << europeanOption.NPV()
291         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
292         << std::setw(widths[3]) << std::left << americanOption.NPV()
293         << std::endl;
294
295     // Monte Carlo Method
296
297     timeSteps = 1;
298
299     method = "MC (crude)";
300     Size mcSeed = 42;
301
302     boost::shared_ptr<PricingEngine> mcengine1;
303     mcengine1 =
304         MakeMCEuropeanEngine<PseudoRandom>().withSteps(timeSteps)
305             .withTolerance(0.02)
306             .withSeed(mcSeed);
307     europeanOption.setPricingEngine(mcengine1);
308     // Real errorEstimate = europeanOption.errorEstimate();
309     std::cout << std::setw(widths[0]) << std::left << method
310         << std::fixed
311         << std::setw(widths[1]) << std::left << europeanOption.NPV()
312         << std::setw(widths[2]) << std::left << "N/A"
313         << std::setw(widths[3]) << std::left << "N/A"
314         << std::endl;
315
316     method = "MC (Sobol)";
317     Size nSamples = 32768; // 2^15
318
319     boost::shared_ptr<PricingEngine> mcengine2;
320     mcengine2 =
321         MakeMCEuropeanEngine<LowDiscrepancy>().withSteps(timeSteps)
322             .withSamples(nSamples);
323     europeanOption.setPricingEngine(mcengine2);
324     std::cout << std::setw(widths[0]) << std::left << method
325         << std::fixed
326         << std::setw(widths[1]) << std::left << europeanOption.NPV()
327         << std::setw(widths[2]) << std::left << "N/A"
328         << std::setw(widths[3]) << std::left << "N/A"
329         << std::endl;
330
331     #if !defined(QL_PATCH_MSVC6) && !defined(QL_PATCH_BORLAND)
332     method = "MC (Longstaff Schwartz)";
333     boost::shared_ptr<PricingEngine> mcengine3;
334     mcengine3 =
335         MakeMCAMericanEngine<PseudoRandom>().withSteps(100)
336             .withAntitheticVariate()
337             .withCalibrationSamples(4096)
338             .withTolerance(0.02)
339             .withSeed(mcSeed);
340     americanOption.setPricingEngine(mcengine3);
341     std::cout << std::setw(widths[0]) << std::left << method
342         << std::fixed
343         << std::setw(widths[1]) << std::left << "N/A"
344         << std::setw(widths[2]) << std::left << "N/A"
345         << std::setw(widths[3]) << std::left << americanOption.NPV()
346         << std::endl;
347     #endif

```

```
348
349     Real seconds = timer.elapsed();
350     Integer hours = int(seconds/3600);
351     seconds -= hours * 3600;
352     Integer minutes = int(seconds/60);
353     seconds -= minutes * 60;
354     std::cout << " \nRun completed in ";
355     if (hours > 0)
356         std::cout << hours << " h ";
357     if (hours > 0 || minutes > 0)
358         std::cout << minutes << " m ";
359     std::cout << std::fixed << std::setprecision(0)
360         << seconds << " s\n" << std::endl;
361
362     return 0;
363 } catch (std::exception& e) {
364     std::cout << e.what() << std::endl;
365     return 1;
366 } catch (...) {
367     std::cout << "unknown error" << std::endl;
368     return 1;
369 }
370 }
```

## 9.5 FRA.cpp

This example evaluates a forward-rate agreement.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 /* This example shows how to set up a term structure and price a simple
21    forward-rate agreement.
22 */
23
24 // the only header you need to use QuantLib
25 #define BOOST_LIB_DIAGNOSTIC
26 # include <ql/quantlib.hpp>
27 #undef BOOST_LIB_DIAGNOSTIC
28
29 #ifdef BOOST_MSVC
30 /* Uncomment the following lines to unmask floating-point
31    exceptions. Warning: unpredictable results can arise...
32
33    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
34    Is there anyone with a definitive word about this?
35 */
36 // #include <float.h>
37 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
38 #endif
39
40 #include <boost/timer.hpp>
41 #include <iostream>
42
43 #define LENGTH(a) (sizeof(a)/sizeof(a[0]))
44
45 using namespace std;
46 using namespace QuantLib;
47
48 #if defined(QL_ENABLE_SESSIONS)
49 namespace QuantLib {
50
51     Integer sessionId() { return 0; }
52
53 }
54 #endif
55
56 int main(int, char* []) {
57
58     try {
59         QL_IO_INIT
60
61         boost::timer timer;
62         std::cout << std::endl;
63
64         /*****
65          *** MARKET DATA ***
66          *****/
67
68         Handle<YieldTermStructure> euriborTermStructure;
69         boost::shared_ptr<Xibor> euribor3m(
70             new Euribor3M(euriborTermStructure));
71
72         Date todaysDate = Date(23, May, 2006);
73         Settings::instance().evaluationDate() = todaysDate;
74
75         Calendar calendar = euribor3m->calendar();
76         Integer fixingDays = euribor3m->settlementDays();
77         Date settlementDate = calendar.advance(todaysDate, fixingDays, Days);
78
79         std::cout << "Today: " << todaysDate.weekday()

```



```

80         << " " << todaysDate << std::endl;
81
82     std::cout << "Settlement date: " << settlementDate.weekday()
83         << " " << settlementDate << std::endl;
84
85
86     // 3 month term FRA quotes (index refers to monthsToStart)
87     Rate threeMonthFraQuote[10];
88
89     threeMonthFraQuote[1]=0.030;
90     threeMonthFraQuote[2]=0.031;
91     threeMonthFraQuote[3]=0.032;
92     threeMonthFraQuote[6]=0.033;
93     threeMonthFraQuote[9]=0.034;
94
95     /*****
96     ***   QUOTES   ***
97     *****/
98
99     // SimpleQuote stores a value which can be manually changed;
100    // other Quote subclasses could read the value from a database
101    // or some kind of data feed.
102
103
104    // FRAs
105    boost::shared_ptr<SimpleQuote> fra1x4Rate(
106        new SimpleQuote(threeMonthFraQuote[1]));
107    boost::shared_ptr<SimpleQuote> fra2x5Rate(
108        new SimpleQuote(threeMonthFraQuote[2]));
109    boost::shared_ptr<SimpleQuote> fra3x6Rate(
110        new SimpleQuote(threeMonthFraQuote[3]));
111    boost::shared_ptr<SimpleQuote> fra6x9Rate(
112        new SimpleQuote(threeMonthFraQuote[6]));
113    boost::shared_ptr<SimpleQuote> fra9x12Rate(
114        new SimpleQuote(threeMonthFraQuote[9]));
115
116    Handle<Quote> h1x4; h1x4.linkTo(fra1x4Rate);
117    Handle<Quote> h2x5; h2x5.linkTo(fra2x5Rate);
118    Handle<Quote> h3x6; h3x6.linkTo(fra3x6Rate);
119    Handle<Quote> h6x9; h6x9.linkTo(fra6x9Rate);
120    Handle<Quote> h9x12; h9x12.linkTo(fra9x12Rate);
121
122    /*****
123    ***   RATE HELPERS   ***
124    *****/
125
126    // RateHelpers are built from the above quotes together with
127    // other instrument dependant infos. Quotes are passed in
128    // relinkable handles which could be relinked to some other
129    // data source later.
130
131    DayCounter fraDayCounter = euribor3m->dayCounter();
132    BusinessDayConvention convention = euribor3m->businessDayConvention();
133
134    boost::shared_ptr<RateHelper> fra1x4(
135        new FraRateHelper(h1x4, 1, 4,
136            fixingDays, calendar, convention,
137            fraDayCounter));
138
139    boost::shared_ptr<RateHelper> fra2x5(
140        new FraRateHelper(h2x5, 2, 5,
141            fixingDays, calendar, convention,
142            fraDayCounter));
143
144    boost::shared_ptr<RateHelper> fra3x6(
145        new FraRateHelper(h3x6, 3, 6,
146            fixingDays, calendar, convention,

```

```

147                                     fraDayCounter));
148
149     boost::shared_ptr<RateHelper> fra6x9(
150         new FraRateHelper(h6x9, 6, 9,
151                             fixingDays, calendar, convention,
152                             fraDayCounter));
153
154     boost::shared_ptr<RateHelper> fra9x12(
155         new FraRateHelper(h9x12, 9, 12,
156                             fixingDays, calendar, convention,
157                             fraDayCounter));
158
159
160     /*****
161     ** CURVE BUILDING **
162     *****/
163
164     // Any DayCounter would be fine.
165     // ActualActual::ISDA ensures that 30 years is 30.0
166     DayCounter termStructureDayCounter =
167         ActualActual(ActualActual::ISDA);
168
169     double tolerance = 1.0e-15;
170
171     // A FRA curve
172     std::vector<boost::shared_ptr<RateHelper> > fraInstruments;
173
174     fraInstruments.push_back(fra1x4);
175     fraInstruments.push_back(fra2x5);
176     fraInstruments.push_back(fra3x6);
177     fraInstruments.push_back(fra6x9);
178     fraInstruments.push_back(fra9x12);
179
180     boost::shared_ptr<YieldTermStructure> fraTermStructure(new
181         PiecewiseFlatForward(settlementDate, fraInstruments,
182                             termStructureDayCounter, tolerance));
183
184
185     // Term structures used for pricing/discounting
186
187     Handle<YieldTermStructure> discountingTermStructure;
188     discountingTermStructure.linkTo(fraTermStructure);
189
190
191     /*****
192     *** construct FRA's ***
193     *****/
194
195     Calendar fraCalendar = euribor3m->calendar();
196     BusinessDayConvention fraBusinessDayConvention =
197         euribor3m->businessDayConvention();
198     Position::Type fraFwdType = Position::Long;
199     Real fraNotional = 100.0;
200     const Integer FraTermMonths = 3;
201     Integer monthsToStart[] = { 1, 2, 3, 6, 9 };
202
203     euriborTermStructure.linkTo(fraTermStructure);
204
205     cout << endl;
206     cout << "Test FRA construction, NPV calculation, and FRA purchase"
207         << endl
208         << endl;
209
210     Size i;
211     for (i=0; i<LENGTH(monthsToStart); i++) {
212
213         Date fraValueDate = fraCalendar.advance(

```

```

214             settlementDate, monthsToStart[i], Months,
215             fraBusinessDayConvention);
216
217     Date fraMaturityDate = fraCalendar.advance(
218         fraValueDate, FraTermMonths, Months,
219         fraBusinessDayConvention);
220
221     Rate fraStrikeRate = threeMonthFraQuote[monthsToStart[i]];
222
223     ForwardRateAgreement myFRA(fraValueDate, fraMaturityDate,
224         fraFwdType, fraStrikeRate,
225         fraNotional, euribor3m,
226         discountingTermStructure);
227
228     cout << "3m Term FRA, Months to Start: "
229         << monthsToStart[i]
230         << endl;
231     cout << "strike FRA rate: "
232         << io::rate(fraStrikeRate)
233         << endl;
234     cout << "FRA 3m forward rate: "
235         << myFRA.forwardRate()
236         << endl;
237     cout << "FRA market quote: "
238         << io::rate(threeMonthFraQuote[monthsToStart[i]])
239         << endl;
240     cout << "FRA spot value: "
241         << myFRA.spotValue()
242         << endl;
243     cout << "FRA forward value: "
244         << myFRA.forwardValue()
245         << endl;
246     cout << "FRA implied Yield: "
247         << myFRA.impliedYield(myFRA.spotValue(),
248             myFRA.forwardValue(),
249             settlementDate,
250             Simple,
251             fraDayCounter)
252         << endl;
253     cout << "market Zero Rate: "
254         << discountingTermStructure->zeroRate(fraMaturityDate,
255             fraDayCounter,
256             Simple)
257         << endl;
258     cout << "FRA NPV [should be zero]: "
259         << myFRA.NPV()
260         << endl
261         << endl;
262 }
263
264
265
266
267
268     cout << endl << endl;
269     cout << "Now take a 100 basis-point upward shift in FRA quotes "
270         << "and examine NPV"
271         << endl
272         << endl;
273
274     const Real BpsShift = 0.01;
275
276     threeMonthFraQuote[1]=0.030+BpsShift;
277     threeMonthFraQuote[2]=0.031+BpsShift;
278     threeMonthFraQuote[3]=0.032+BpsShift;
279     threeMonthFraQuote[6]=0.033+BpsShift;
280     threeMonthFraQuote[9]=0.034+BpsShift;

```

```

281
282     fra1x4Rate->setValue(threeMonthFraQuote[1]);
283     fra2x5Rate->setValue(threeMonthFraQuote[2]);
284     fra3x6Rate->setValue(threeMonthFraQuote[3]);
285     fra6x9Rate->setValue(threeMonthFraQuote[6]);
286     fra9x12Rate->setValue(threeMonthFraQuote[9]);
287
288
289     for (i=0; i<LENGTH(monthsToStart); i++) {
290
291         Date fraValueDate = fraCalendar.advance(
292             settlementDate, monthsToStart[i], Months,
293             fraBusinessDayConvention);
294
295         Date fraMaturityDate = fraCalendar.advance(
296             fraValueDate, FraTermMonths, Months,
297             fraBusinessDayConvention);
298
299         Rate fraStrikeRate =
300             threeMonthFraQuote[monthsToStart[i]] - BpsShift;
301
302         ForwardRateAgreement myFRA(fraValueDate, fraMaturityDate,
303             fraFwdType, fraStrikeRate,
304             fraNotional, euribor3m,
305             discountingTermStructure);
306
307         cout << "3m Term FRA, 100 notional, Months to Start = "
308             << monthsToStart[i]
309             << endl;
310         cout << "strike FRA rate: "
311             << io::rate(fraStrikeRate)
312             << endl;
313         cout << "FRA 3m forward rate: "
314             << myFRA.forwardRate()
315             << endl;
316         cout << "FRA market quote: "
317             << io::rate(threeMonthFraQuote[monthsToStart[i]])
318             << endl;
319         cout << "FRA spot value: "
320             << myFRA.spotValue()
321             << endl;
322         cout << "FRA forward value: "
323             << myFRA.forwardValue()
324             << endl;
325         cout << "FRA implied Yield: "
326             << myFRA.impliedYield(myFRA.spotValue(),
327                 myFRA.forwardValue(),
328                 settlementDate,
329                 Simple,
330                 fraDayCounter)
331             << endl;
332         cout << "market Zero Rate: "
333             << discountingTermStructure->zeroRate(fraMaturityDate,
334                 fraDayCounter,
335                 Simple)
336             << endl;
337         cout << "FRA NPV [should be positive]: "
338             << myFRA.NPV()
339             << endl;
340             << endl;
341     }
342
343     Real seconds = timer.elapsed();
344     Integer hours = int(seconds/3600);
345     seconds -= hours * 3600;
346     Integer minutes = int(seconds/60);
347     seconds -= minutes * 60;

```

```
348         cout << " \nRun completed in ";
349         if (hours > 0)
350             cout << hours << " h ";
351         if (hours > 0 || minutes > 0)
352             cout << minutes << " m ";
353         cout << fixed << setprecision(0)
354             << seconds << " s\n" << endl;
355
356         return 0;
357
358     } catch (exception& e) {
359         cout << e.what() << endl;
360         return 1;
361     } catch (...) {
362         cout << "unknown error" << endl;
363         return 1;
364     }
365 }
366
```

## 9.6 Replication.cpp

This example uses the CompositeInstrument class to perform static replication of a down-and-out barrier option.

```

1 /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 /* This example showcases the CompositeInstrument class. Such class
21    is used to build a static replication of a down-and-out barrier
22    option, as outlined in Section 10.2 of Mark Joshi's "The Concepts
23    and Practice of Mathematical Finance" to which we refer the
24    reader.
25 */
26
27 // the only header you need to use QuantLib
28 #define BOOST_LIB_DIAGNOSTIC
29 # include <ql/quantlib.hpp>
30 #undef BOOST_LIB_DIAGNOSTIC
31
32 #ifdef BOOST_MSVC
33 /* Uncomment the following lines to unmask floating-point
34    exceptions. Warning: unpredictable results can arise...
35
36    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
37    Is there anyone with a definitive word about this?
38 */
39 // #include <float.h>
40 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
41 #endif
42
43 #include <boost/timer.hpp>
44 #include <iostream>
45 #include <iomanip>
46
47 using namespace QuantLib;
48
49 #if defined(QL_ENABLE_SESSIONS)
50 namespace QuantLib {
51     Integer sessionId() { return 0; }
52 }
53 #endif
54
55 int main(int, char* [])
56 {
57     try {
58         QL_IO_INIT
59
60         boost::timer timer;
61         std::cout << std::endl;
62
63         Date today(29, May, 2006);
64         Settings::instance().evaluationDate() = today;
65
66         // the option to replicate
67         Barrier::Type barrierType = Barrier::DownOut;
68         Real barrier = 70.0;
69         Real rebate = 0.0;
70         Option::Type type = Option::Put;
71         Real underlyingValue = 100.0;
72         boost::shared_ptr<SimpleQuote> underlying(
73             new SimpleQuote(underlyingValue));
74
75         Real strike = 100.0;
76         boost::shared_ptr<SimpleQuote> riskFreeRate(new SimpleQuote(0.04));
77         boost::shared_ptr<SimpleQuote> volatility(new SimpleQuote(0.20));

```

```

79     Date maturity = today + 1*Years;
80
81     std::cout << std::endl ;
82
83     // write column headings
84     Size widths[] = { 45, 15, 15 };
85     Size totalWidth = widths[0]+widths[1]+widths[2];
86     std::string rule(totalWidth, '-'), dblrule(totalWidth, '=');
87
88     std::cout << dblrule << std::endl;
89     std::cout << "Initial market conditions" << std::endl;
90     std::cout << dblrule << std::endl;
91     std::cout << std::setw(widths[0]) << std::left << "Option"
92               << std::setw(widths[1]) << std::left << "NPV"
93               << std::setw(widths[2]) << std::left << "Error"
94               << std::endl;
95     std::cout << rule << std::endl;
96
97     // bootstrap the yield/vol curves
98     DayCounter dayCounter = Actual365Fixed();
99     Handle<Quote> h1(riskFreeRate);
100    Handle<Quote> h2(volatility);
101    Handle<YieldTermStructure> flatRate(
102        boost::shared_ptr<YieldTermStructure>(
103            new FlatForward(0, NullCalendar(),
104                            h1, dayCounter)));
105    Handle<BlackVolTermStructure> flatVol(
106        boost::shared_ptr<BlackVolTermStructure>(
107            new BlackConstantVol(0, NullCalendar(),
108                                  h2, dayCounter)));
109
110    // instantiate the option
111    boost::shared_ptr<Exercise> exercise(
112        new EuropeanExercise(maturity));
113    boost::shared_ptr<StrikedTypePayoff> payoff(
114        new PlainVanillaPayoff(type, strike));
115
116    boost::shared_ptr<StochasticProcess> stochasticProcess(
117        new BlackScholesProcess(Handle<Quote>(underlying),
118                                  flatRate, flatVol));
119
120    BarrierOption referenceOption(barrierType, barrier, rebate,
121                                  stochasticProcess, payoff, exercise);
122
123    Real referenceValue = referenceOption.NPV();
124
125    std::cout << std::setw(widths[0]) << std::left
126              << "Original barrier option"
127              << std::fixed
128              << std::setw(widths[1]) << std::left << referenceValue
129              << std::setw(widths[2]) << std::left << "N/A"
130              << std::endl;
131
132    // Replicating portfolios
133    CompositeInstrument portfolio1, portfolio2, portfolio3;
134
135    // Final payoff first (the same for all portfolios):
136    // as shown in Joshi, a put struck at K...
137    boost::shared_ptr<Instrument> put1(
138        new EuropeanOption(stochasticProcess, payoff, exercise));
139    portfolio1.add(put1);
140    portfolio2.add(put1);
141    portfolio3.add(put1);
142    // ...minus a digital put struck at B of notional K-B...
143    boost::shared_ptr<StrikedTypePayoff> digitalPayoff(
144        new CashOrNothingPayoff(Option::Put, barrier, 1.0));
145    boost::shared_ptr<Instrument> digitalPut(

```

```

146         new EuropeanOption(stochasticProcess, digitalPayoff, exercise));
147 portfolio1.subtract(digitalPut, strike-barrier);
148 portfolio2.subtract(digitalPut, strike-barrier);
149 portfolio3.subtract(digitalPut, strike-barrier);
150 // ...minus a put option struck at B.
151 boost::shared_ptr<StrikedTypePayoff> lowerPayoff(
152     new PlainVanillaPayoff(Option::Put, barrier));
153 boost::shared_ptr<Instrument> put2(
154     new EuropeanOption(stochasticProcess, lowerPayoff, exercise));
155 portfolio1.subtract(put2);
156 portfolio2.subtract(put2);
157 portfolio3.subtract(put2);
158
159 // Now we use puts struck at B to kill the value of the
160 // portfolio on a number of points (B,t). For the first
161 // portfolio, we'll use 12 dates at one-month's distance.
162 Integer i;
163 for (i=12; i>=1; i--) {
164     // First, we instantiate the option...
165     Date innerMaturity = today + i*Months;
166     boost::shared_ptr<Exercise> innerExercise(
167         new EuropeanExercise(innerMaturity));
168     boost::shared_ptr<StrikedTypePayoff> innerPayoff(
169         new PlainVanillaPayoff(Option::Put, barrier));
170     boost::shared_ptr<Instrument> putn(
171         new EuropeanOption(stochasticProcess,
172             innerPayoff, innerExercise));
173     // ...second, we evaluate the current portfolio and the
174     // latest put at (B,t)...
175     Date killDate = today + (i-1)*Months;
176     Settings::instance().evaluationDate() = killDate;
177     underlying->setBarrier(killDate);
178     Real portfolioValue = portfolio1.NPV();
179     Real putValue = putn->NPV();
180     // ...finally, we estimate the notional that kills the
181     // portfolio value at that point...
182     Real notional = portfolioValue/putValue;
183     // ...and we subtract from the portfolio a put with such
184     // notional.
185     portfolio1.subtract(putn, notional);
186 }
187 // The portfolio being complete, we return to today's market...
188 Settings::instance().evaluationDate() = today;
189 underlying->setBarrier(killDate);
190 // ...and output the value.
191 Real portfolioValue = portfolio1.NPV();
192 Real error = portfolioValue - referenceValue;
193 std::cout << std::setw(widths[0]) << std::left
194     << "Replicating portfolio (12 dates)"
195     << std::fixed
196     << std::setw(widths[1]) << std::left << portfolioValue
197     << std::setw(widths[2]) << std::left << error
198     << std::endl;
199
200 // For the second portfolio, we'll use 26 dates at two-weeks'
201 // distance.
202 for (i=52; i>=2; i-=2) {
203     // Same as above.
204     Date innerMaturity = today + i*Weeks;
205     boost::shared_ptr<Exercise> innerExercise(
206         new EuropeanExercise(innerMaturity));
207     boost::shared_ptr<StrikedTypePayoff> innerPayoff(
208         new PlainVanillaPayoff(Option::Put, barrier));
209     boost::shared_ptr<Instrument> putn(
210         new EuropeanOption(stochasticProcess,
211             innerPayoff, innerExercise));
212     Date killDate = today + (i-2)*Weeks;

```



```

213         Settings::instance().evaluationDate() = killDate;
214         underlying->setValue(barrier);
215         Real portfolioValue = portfolio2.NPV();
216         Real putValue = putn->NPV();
217         Real notional = portfolioValue/putValue;
218         portfolio2.subtract(putn, notional);
219     }
220     Settings::instance().evaluationDate() = today;
221     underlying->setValue(underlyingValue);
222     portfolioValue = portfolio2.NPV();
223     error = portfolioValue - referenceValue;
224     std::cout << std::setw(widths[0]) << std::left
225         << "Replicating portfolio (26 dates)"
226         << std::fixed
227         << std::setw(widths[1]) << std::left << portfolioValue
228         << std::setw(widths[2]) << std::left << error
229         << std::endl;
230
231     // For the third portfolio, we'll use 52 dates at one-week's
232     // distance.
233     for (i=52; i>=1; i--) {
234         // Same as above.
235         Date innerMaturity = today + i*Weeks;
236         boost::shared_ptr<Exercise> innerExercise(
237             new EuropeanExercise(innerMaturity));
238         boost::shared_ptr<StrikedTypePayoff> innerPayoff(
239             new PlainVanillaPayoff(Option::Put, barrier));
240         boost::shared_ptr<Instrument> putn(
241             new EuropeanOption(stochasticProcess,
242                 innerPayoff, innerExercise));
243         Date killDate = today + (i-1)*Weeks;
244         Settings::instance().evaluationDate() = killDate;
245         underlying->setValue(barrier);
246         Real portfolioValue = portfolio3.NPV();
247         Real putValue = putn->NPV();
248         Real notional = portfolioValue/putValue;
249         portfolio3.subtract(putn, notional);
250     }
251     Settings::instance().evaluationDate() = today;
252     underlying->setValue(underlyingValue);
253     portfolioValue = portfolio3.NPV();
254     error = portfolioValue - referenceValue;
255     std::cout << std::setw(widths[0]) << std::left
256         << "Replicating portfolio (52 dates)"
257         << std::fixed
258         << std::setw(widths[1]) << std::left << portfolioValue
259         << std::setw(widths[2]) << std::left << error
260         << std::endl;
261
262     // Now we modify the market condition to see whether the
263     // replication holds. First, we change the underlying value so
264     // that the option is out of the money.
265     std::cout << dblrule << std::endl;
266     std::cout << "Modified market conditions: out of the money"
267         << std::endl;
268     std::cout << dblrule << std::endl;
269     std::cout << std::setw(widths[0]) << std::left << "Option"
270         << std::setw(widths[1]) << std::left << "NPV"
271         << std::setw(widths[2]) << std::left << "Error"
272         << std::endl;
273     std::cout << rule << std::endl;
274
275     underlying->setValue(110.0);
276
277     referenceValue = referenceOption.NPV();
278     std::cout << std::setw(widths[0]) << std::left
279         << "Original barrier option"

```

```

280         << std::fixed
281         << std::setw(widths[1]) << std::left << referenceValue
282         << std::setw(widths[2]) << std::left << "N/A"
283         << std::endl;
284     portfolioValue = portfolio1.NPV();
285     error = portfolioValue - referenceValue;
286     std::cout << std::setw(widths[0]) << std::left
287     << "Replicating portfolio (12 dates)"
288     << std::fixed
289     << std::setw(widths[1]) << std::left << portfolioValue
290     << std::setw(widths[2]) << std::left << error
291     << std::endl;
292     portfolioValue = portfolio2.NPV();
293     error = portfolioValue - referenceValue;
294     std::cout << std::setw(widths[0]) << std::left
295     << "Replicating portfolio (26 dates)"
296     << std::fixed
297     << std::setw(widths[1]) << std::left << portfolioValue
298     << std::setw(widths[2]) << std::left << error
299     << std::endl;
300     portfolioValue = portfolio3.NPV();
301     error = portfolioValue - referenceValue;
302     std::cout << std::setw(widths[0]) << std::left
303     << "Replicating portfolio (52 dates)"
304     << std::fixed
305     << std::setw(widths[1]) << std::left << portfolioValue
306     << std::setw(widths[2]) << std::left << error
307     << std::endl;
308
309     // Next, we change the underlying value so that the option is
310     // in the money.
311     std::cout << dblrule << std::endl;
312     std::cout << "Modified market conditions: in the money" << std::endl;
313     std::cout << dblrule << std::endl;
314     std::cout << std::setw(widths[0]) << std::left << "Option"
315     << std::setw(widths[1]) << std::left << "NPV"
316     << std::setw(widths[2]) << std::left << "Error"
317     << std::endl;
318     std::cout << rule << std::endl;
319
320     underlying->setValue(90.0);
321
322     referenceValue = referenceOption.NPV();
323     std::cout << std::setw(widths[0]) << std::left
324     << "Original barrier option"
325     << std::fixed
326     << std::setw(widths[1]) << std::left << referenceValue
327     << std::setw(widths[2]) << std::left << "N/A"
328     << std::endl;
329     portfolioValue = portfolio1.NPV();
330     error = portfolioValue - referenceValue;
331     std::cout << std::setw(widths[0]) << std::left
332     << "Replicating portfolio (12 dates)"
333     << std::fixed
334     << std::setw(widths[1]) << std::left << portfolioValue
335     << std::setw(widths[2]) << std::left << error
336     << std::endl;
337     portfolioValue = portfolio2.NPV();
338     error = portfolioValue - referenceValue;
339     std::cout << std::setw(widths[0]) << std::left
340     << "Replicating portfolio (26 dates)"
341     << std::fixed
342     << std::setw(widths[1]) << std::left << portfolioValue
343     << std::setw(widths[2]) << std::left << error
344     << std::endl;
345     portfolioValue = portfolio3.NPV();
346     error = portfolioValue - referenceValue;

```

```
347         std::cout << std::setw(widths[0]) << std::left
348             << "Replicating portfolio (52 dates)"
349             << std::fixed
350             << std::setw(widths[1]) << std::left << portfolioValue
351             << std::setw(widths[2]) << std::left << error
352             << std::endl;
353
354         // Finally, a word of warning for those (shame on them) who
355         // run the example but do not read the code.
356         std::cout << dblrule << std::endl;
357         std::cout
358             << std::endl
359             << "The replication seems to be less robust when volatility and \n"
360             << "risk-free rate are changed. Feel free to experiment with \n"
361             << "the example and contribute a patch if you spot any errors."
362             << std::endl;
363
364         Real seconds = timer.elapsed();
365         Integer hours = int(seconds/3600);
366         seconds -= hours * 3600;
367         Integer minutes = int(seconds/60);
368         seconds -= minutes * 60;
369         std::cout << " \nRun completed in ";
370         if (hours > 0)
371             std::cout << hours << " h ";
372         if (hours > 0 || minutes > 0)
373             std::cout << minutes << " m ";
374         std::cout << std::fixed << std::setprecision(0)
375             << seconds << " s\n" << std::endl;
376
377         return 0;
378     } catch (std::exception& e) {
379         std::cout << e.what() << std::endl;
380         return 1;
381     } catch (...) {
382         std::cout << "unknown error" << std::endl;
383         return 1;
384     }
385 }
```

## 9.7 Repo.cpp

This example evaluates a repo on a fixed-coupon bond.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 /* a Repo calculation done using the FixedCouponBondForward class
21    cf. aaBondFwd() repo example at
22    http://www.fincad.com/support/developerFunc/mathref/BFWD.htm
23
24    This repo is set up to use the repo rate to do all discounting
25    (including the underlying bond income). Forward delivery price is
26    also obtained using this repo rate. All this is done by supplying
27    the FixedCouponBondForward constructor with a flat repo
28    YieldTermStructure.
29 */
30
31 // the only header you need to use QuantLib
32 #define BOOST_LIB_DIAGNOSTIC
33 # include <ql/quantlib.hpp>
34 #undef BOOST_LIB_DIAGNOSTIC
35
36 #ifdef BOOST_MSVC
37 /* Uncomment the following lines to unmask floating-point
38    exceptions. Warning: unpredictable results can arise...
39
40    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
41    Is there anyone with a definitive word about this?
42 */
43 // #include <float.h>
44 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
45 #endif
46
47 #include <boost/timer.hpp>
48 #include <iostream>
49
50 using namespace std;
51 using namespace QuantLib;
52
53 #if defined(QL_ENABLE_SESSIONS)
54 namespace QuantLib {
55     Integer sessionId() { return 0; }
56 }
57 #endif
58
59 int main(int, char* []) {
60
61     try {
62         QL_IO_INIT
63
64         boost::timer timer;
65         std::cout << std::endl;
66
67         Date repoSettlementDate(14, February, 2000);
68         Date repoDeliveryDate(15, August, 2000);
69         Rate repoRate = 0.05;
70         DayCounter repoDayCountConvention = Actual360();
71         Integer repoSettlementDays = 0;
72         Compounding repoCompounding = Simple;
73         Frequency repoCompoundFreq = Annual;
74
75         // assume a ten year bond- this is irrelevant
76         Date bondIssueDate(15, September, 1995);
77         Date bondDatedDate(15, September, 1995);

```

```

80     Date bondMaturityDate(15,September,2005);
81     Real bondCoupon = 0.08;
82     Frequency bondCouponFrequency = Semiannual;
83     // unknown what calendar fincad is using
84     Calendar bondCalendar = NullCalendar();
85     DayCounter bondDayCountConvention = Thirty360(Thirty360::BondBasis);
86     // unknown what fincad is using. this may affect accrued calculation
87     Integer bondSettlementDays = 0;
88     BusinessDayConvention bondBusinessDayConvention = Unadjusted;
89     Real bondCleanPrice = 89.97693786;
90     Real bondRedemption = 100.0;
91     Real faceAmount = 100.0;
92
93
94     Settings::instance().evaluationDate() = repoSettlementDate;
95
96     Handle<YieldTermStructure> bondCurve;
97     bondCurve.linkTo(boost::shared_ptr<YieldTermStructure>(
98         new FlatForward(repoSettlementDate,
99             .01, // dummy rate
100             bondDayCountConvention,
101             Compounded,
102             bondCouponFrequency)));
103
104
105     boost::shared_ptr<FixedCouponBond> bond(
106         new FixedCouponBond(faceAmount,
107             bondIssueDate,
108             bondDatedDate,
109             bondMaturityDate,
110             bondSettlementDays,
111             std::vector<Rate>(1,bondCoupon),
112             bondCouponFrequency,
113             bondCalendar,
114             bondDayCountConvention,
115             bondBusinessDayConvention,
116             bondBusinessDayConvention,
117             bondRedemption,
118             bondCurve));
119
120
121     bondCurve.linkTo(boost::shared_ptr<YieldTermStructure> (
122         new FlatForward(repoSettlementDate,
123             bond->yield(bondCleanPrice,Compounded),
124             bondDayCountConvention,
125             Compounded,
126             bondCouponFrequency)));
127
128     Position::Type fwdType = Position::Long;
129     double dummyStrike = 91.5745;
130
131     Handle<YieldTermStructure> repoCurve;
132     repoCurve.linkTo(boost::shared_ptr<YieldTermStructure> (
133         new FlatForward(repoSettlementDate,
134             repoRate,
135             repoDayCountConvention,
136             repoCompounding,
137             repoCompoundFreq)));
138
139
140     FixedCouponBondForward bondFwd(repoSettlementDate,
141         repoDeliveryDate,
142         fwdType,
143         dummyStrike,
144         repoSettlementDays,
145         repoDayCountConvention,
146         bondCalendar,

```

```

147                                     bondBusinessDayConvention,
148                                     bond,
149                                     repoCurve,
150                                     repoCurve);
151
152
153     cout << "Underlying bond clean price: "
154           << bond->cleanPrice()
155           << endl;
156     cout << "Underlying bond dirty price: "
157           << bond->dirtyPrice()
158           << endl;
159     cout << "Underlying bond accrued at settlement: "
160           << bond->accruedAmount(repoSettlementDate)
161           << endl;
162     cout << "Underlying bond accrued at delivery: "
163           << bond->accruedAmount(repoDeliveryDate)
164           << endl;
165     cout << "Underlying bond spot income: "
166           << bondFwd.spotIncome(repoCurve)
167           << endl;
168     cout << "Underlying bond fwd income: "
169           << bondFwd.spotIncome(repoCurve)/
170           << repoCurve->discount(repoDeliveryDate)
171           << endl;
172     cout << "Repo strike: "
173           << dummyStrike
174           << endl;
175     cout << "Repo NPV: "
176           << bondFwd.NPV()
177           << endl;
178     cout << "Repo clean forward price: "
179           << bondFwd.cleanForwardPrice()
180           << endl;
181     cout << "Repo dirty forward price: "
182           << bondFwd.forwardPrice()
183           << endl;
184     cout << "Repo implied yield: "
185           << bondFwd.impliedYield(bond->dirtyPrice(),
186                                   dummyStrike,
187                                   repoSettlementDate,
188                                   repoCompounding,
189                                   repoDayCountConvention)
190           << endl;
191     cout << "Market repo rate: "
192           << repoCurve->zeroRate(repoDeliveryDate,
193                                   repoDayCountConvention,
194                                   repoCompounding,
195                                   repoCompoundFreq)
196           << endl
197           << endl;
198
199     cout << "Compare with example given at \n"
200           << "http://www.fincad.com/support/developerFunc/mathref/BFWD.htm"
201           << endl;
202     cout << "Clean forward price = 88.2408"
203           << endl
204           << endl;
205     cout << "In that example, it is unknown what bond calendar they are\n"
206           << "using, as well as settlement Days. For that reason, I have\n"
207           << "made the simplest possible assumptions here: NullCalendar\n"
208           << "and 0 settlement days."
209           << endl;
210
211
212     Real seconds = timer.elapsed();
213     Integer hours = int(seconds/3600);

```

```
214         seconds -= hours * 3600;
215         Integer minutes = int(seconds/60);
216         seconds -= minutes * 60;
217         cout << " \nRun completed in ";
218         if (hours > 0)
219             cout << hours << " h ";
220         if (hours > 0 || minutes > 0)
221             cout << minutes << " m ";
222         cout << fixed << setprecision(0)
223             << seconds << " s\n" << endl;
224
225         return 0;
226
227     } catch (exception& e) {
228         cout << e.what() << endl;
229         return 1;
230     } catch (...) {
231         cout << "unknown error" << endl;
232         return 1;
233     }
234 }
235
```

## 9.8 swapvaluation.cpp

This is an example of using the QuantLib Term Structure for pricing a simple swap.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
22 /* This example shows how to set up a Term Structure and then price a simple
23    swap.
24 */
25
26 // the only header you need to use QuantLib
27 #define BOOST_LIB_DIAGNOSTIC
28 # include <ql/quantlib.hpp>
29 #undef BOOST_LIB_DIAGNOSTIC
30
31 #ifdef BOOST_MSVC
32 /* Uncomment the following lines to unmask floating-point
33    exceptions. Warning: unpredictable results can arise...
34
35    See http://www.wilmott.com/messageview.cfm?catid=10&threadid=9481
36    Is there anyone with a definitive word about this?
37 */
38 // #include <float.h>
39 // namespace { unsigned int u = _controlfp(_EM_INEXACT, _MCW_EM); }
40 #endif
41
42 #include <boost/timer.hpp>
43 #include <iostream>
44 #include <iomanip>
45
46 using namespace QuantLib;
47
48 #if defined(QL_ENABLE_SESSIONS)
49 namespace QuantLib {
50
51     Integer sessionId() { return 0; }
52
53 }
54 #endif
55
56
57 int main(int, char* [])
58 {
59     try {
60         QL_IO_INIT
61
62         boost::timer timer;
63         std::cout << std::endl;
64
65         /*****
66          *** MARKET DATA ***
67          *****/
68
69         Calendar calendar = TARGET();
70         // uncommenting the following line generates an error
71         // calendar = Tokyo();
72         Date settlementDate(22, September, 2004);
73         // must be a business day
74         settlementDate = calendar.adjust(settlementDate);
75
76         Integer fixingDays = 2;
77         Date todaysDate = calendar.advance(settlementDate, -fixingDays, Days);
78         // nothing to do with Date::todaysDate
79         Settings::instance().evaluationDate() = todaysDate;
80
81

```



```

82     todaysDate = Settings::instance().evaluationDate();
83     std::cout << "Today: " << todaysDate.weekday()
84               << ", " << todaysDate << std::endl;
85
86     std::cout << "Settlement date: " << settlementDate.weekday()
87               << ", " << settlementDate << std::endl;
88
89     // deposits
90     Rate d1wQuote=0.0382;
91     Rate d1mQuote=0.0372;
92     Rate d3mQuote=0.0363;
93     Rate d6mQuote=0.0353;
94     Rate d9mQuote=0.0348;
95     Rate d1yQuote=0.0345;
96     // FRAs
97     Rate fra3x6Quote=0.037125;
98     Rate fra6x9Quote=0.037125;
99     Rate fra6x12Quote=0.037125;
100    // futures
101    Real fut1Quote=96.2875;
102    Real fut2Quote=96.7875;
103    Real fut3Quote=96.9875;
104    Real fut4Quote=96.6875;
105    Real fut5Quote=96.4875;
106    Real fut6Quote=96.3875;
107    Real fut7Quote=96.2875;
108    Real fut8Quote=96.0875;
109    // swaps
110    Rate s2yQuote=0.037125;
111    Rate s3yQuote=0.0398;
112    Rate s5yQuote=0.0443;
113    Rate s10yQuote=0.05165;
114    Rate s15yQuote=0.055175;
115
116
117    /*****
118     ***   QUOTES   ***
119     *****/
120
121    // SimpleQuote stores a value which can be manually changed;
122    // other Quote subclasses could read the value from a database
123    // or some kind of data feed.
124
125    // deposits
126    boost::shared_ptr<Quote> d1wRate(new SimpleQuote(d1wQuote));
127    boost::shared_ptr<Quote> d1mRate(new SimpleQuote(d1mQuote));
128    boost::shared_ptr<Quote> d3mRate(new SimpleQuote(d3mQuote));
129    boost::shared_ptr<Quote> d6mRate(new SimpleQuote(d6mQuote));
130    boost::shared_ptr<Quote> d9mRate(new SimpleQuote(d9mQuote));
131    boost::shared_ptr<Quote> d1yRate(new SimpleQuote(d1yQuote));
132    // FRAs
133    boost::shared_ptr<Quote> fra3x6Rate(new SimpleQuote(fra3x6Quote));
134    boost::shared_ptr<Quote> fra6x9Rate(new SimpleQuote(fra6x9Quote));
135    boost::shared_ptr<Quote> fra6x12Rate(new SimpleQuote(fra6x12Quote));
136    // futures
137    boost::shared_ptr<Quote> fut1Price(new SimpleQuote(fut1Quote));
138    boost::shared_ptr<Quote> fut2Price(new SimpleQuote(fut2Quote));
139    boost::shared_ptr<Quote> fut3Price(new SimpleQuote(fut3Quote));
140    boost::shared_ptr<Quote> fut4Price(new SimpleQuote(fut4Quote));
141    boost::shared_ptr<Quote> fut5Price(new SimpleQuote(fut5Quote));
142    boost::shared_ptr<Quote> fut6Price(new SimpleQuote(fut6Quote));
143    boost::shared_ptr<Quote> fut7Price(new SimpleQuote(fut7Quote));
144    boost::shared_ptr<Quote> fut8Price(new SimpleQuote(fut8Quote));
145    // swaps
146    boost::shared_ptr<Quote> s2yRate(new SimpleQuote(s2yQuote));
147    boost::shared_ptr<Quote> s3yRate(new SimpleQuote(s3yQuote));
148    boost::shared_ptr<Quote> s5yRate(new SimpleQuote(s5yQuote));

```

```

149     boost::shared_ptr<Quote> s10yRate(new SimpleQuote(s10yQuote));
150     boost::shared_ptr<Quote> s15yRate(new SimpleQuote(s15yQuote));
151
152
153     /*****
154     ***   RATE HELPERS   ***
155     *****/
156
157     // RateHelpers are built from the above quotes together with
158     // other instrument dependant infos. Quotes are passed in
159     // relinkable handles which could be relinked to some other
160     // data source later.
161
162     // deposits
163     DayCounter depositDayCounter = Actual360();
164
165     boost::shared_ptr<RateHelper> d1w(new DepositRateHelper(
166         Handle<Quote>(d1wRate),
167         1*Weeks, fixingDays,
168         calendar, ModifiedFollowing, depositDayCounter));
169     boost::shared_ptr<RateHelper> d1m(new DepositRateHelper(
170         Handle<Quote>(d1mRate),
171         1*Months, fixingDays,
172         calendar, ModifiedFollowing, depositDayCounter));
173     boost::shared_ptr<RateHelper> d3m(new DepositRateHelper(
174         Handle<Quote>(d3mRate),
175         3*Months, fixingDays,
176         calendar, ModifiedFollowing, depositDayCounter));
177     boost::shared_ptr<RateHelper> d6m(new DepositRateHelper(
178         Handle<Quote>(d6mRate),
179         6*Months, fixingDays,
180         calendar, ModifiedFollowing, depositDayCounter));
181     boost::shared_ptr<RateHelper> d9m(new DepositRateHelper(
182         Handle<Quote>(d9mRate),
183         9*Months, fixingDays,
184         calendar, ModifiedFollowing, depositDayCounter));
185     boost::shared_ptr<RateHelper> d1y(new DepositRateHelper(
186         Handle<Quote>(d1yRate),
187         1*Years, fixingDays,
188         calendar, ModifiedFollowing, depositDayCounter));
189
190
191     // setup FRAs
192     boost::shared_ptr<RateHelper> fra3x6(new FraRateHelper(
193         Handle<Quote>(fra3x6Rate),
194         3, 6, fixingDays, calendar, ModifiedFollowing,
195         depositDayCounter));
196     boost::shared_ptr<RateHelper> fra6x9(new FraRateHelper(
197         Handle<Quote>(fra6x9Rate),
198         6, 9, fixingDays, calendar, ModifiedFollowing,
199         depositDayCounter));
200     boost::shared_ptr<RateHelper> fra6x12(new FraRateHelper(
201         Handle<Quote>(fra6x12Rate),
202         6, 12, fixingDays, calendar, ModifiedFollowing,
203         depositDayCounter));
204
205
206     // setup futures
207     Rate convexityAdjustment = 0.0;
208     Integer futMonths = 3;
209     Date imm = Date::nextIMMdate(settlementDate);
210     boost::shared_ptr<RateHelper> fut1(new FuturesRateHelper(
211         Handle<Quote>(fut1Price),
212         imm,
213         futMonths, calendar, ModifiedFollowing,
214         depositDayCounter, convexityAdjustment));
215     imm = Date::nextIMMdate(imm+1);

```

```

216         boost::shared_ptr<RateHelper> fut2(new FuturesRateHelper(
217             Handle<Quote>(fut1Price),
218             imm,
219             futMonths, calendar, ModifiedFollowing,
220             depositDayCounter, convexityAdjustment));
221         imm = Date::nextIMMdate(imm+1);
222         boost::shared_ptr<RateHelper> fut3(new FuturesRateHelper(
223             Handle<Quote>(fut1Price),
224             imm,
225             futMonths, calendar, ModifiedFollowing,
226             depositDayCounter, convexityAdjustment));
227         imm = Date::nextIMMdate(imm+1);
228         boost::shared_ptr<RateHelper> fut4(new FuturesRateHelper(
229             Handle<Quote>(fut1Price),
230             imm,
231             futMonths, calendar, ModifiedFollowing,
232             depositDayCounter, convexityAdjustment));
233         imm = Date::nextIMMdate(imm+1);
234         boost::shared_ptr<RateHelper> fut5(new FuturesRateHelper(
235             Handle<Quote>(fut1Price),
236             imm,
237             futMonths, calendar, ModifiedFollowing,
238             depositDayCounter, convexityAdjustment));
239         imm = Date::nextIMMdate(imm+1);
240         boost::shared_ptr<RateHelper> fut6(new FuturesRateHelper(
241             Handle<Quote>(fut1Price),
242             imm,
243             futMonths, calendar, ModifiedFollowing,
244             depositDayCounter, convexityAdjustment));
245         imm = Date::nextIMMdate(imm+1);
246         boost::shared_ptr<RateHelper> fut7(new FuturesRateHelper(
247             Handle<Quote>(fut1Price),
248             imm,
249             futMonths, calendar, ModifiedFollowing,
250             depositDayCounter, convexityAdjustment));
251         imm = Date::nextIMMdate(imm+1);
252         boost::shared_ptr<RateHelper> fut8(new FuturesRateHelper(
253             Handle<Quote>(fut1Price),
254             imm,
255             futMonths, calendar, ModifiedFollowing,
256             depositDayCounter, convexityAdjustment));
257
258
259         // setup swaps
260         Frequency swFixedLegFrequency = Annual;
261         BusinessDayConvention swFixedLegConvention = Unadjusted;
262         DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
263         boost::shared_ptr<Xibor> swFloatingLegIndex(new Euribor6M);
264
265         boost::shared_ptr<RateHelper> s2y(new SwapRateHelper(
266             Handle<Quote>(s2yRate),
267             2*Years, fixingDays,
268             calendar, swFixedLegFrequency,
269             swFixedLegConvention, swFixedLegDayCounter,
270             swFloatingLegIndex));
271         boost::shared_ptr<RateHelper> s3y(new SwapRateHelper(
272             Handle<Quote>(s3yRate),
273             3*Years, fixingDays,
274             calendar, swFixedLegFrequency,
275             swFixedLegConvention, swFixedLegDayCounter,
276             swFloatingLegIndex));
277         boost::shared_ptr<RateHelper> s5y(new SwapRateHelper(
278             Handle<Quote>(s5yRate),
279             5*Years, fixingDays,
280             calendar, swFixedLegFrequency,
281             swFixedLegConvention, swFixedLegDayCounter,
282             swFloatingLegIndex));

```

```

283     boost::shared_ptr<RateHelper> s10y(new SwapRateHelper(
284         Handle<Quote>(s10yRate),
285         10*Years, fixingDays,
286         calendar, swFixedLegFrequency,
287         swFixedLegConvention, swFixedLegDayCounter,
288         swFloatingLegIndex));
289     boost::shared_ptr<RateHelper> s15y(new SwapRateHelper(
290         Handle<Quote>(s15yRate),
291         15*Years, fixingDays,
292         calendar, swFixedLegFrequency,
293         swFixedLegConvention, swFixedLegDayCounter,
294         swFloatingLegIndex));
295
296
297     /*****
298     ** CURVE BUILDING **
299     *****/
300
301     // Any DayCounter would be fine.
302     // ActualActual::ISDA ensures that 30 years is 30.0
303     DayCounter termStructureDayCounter =
304         ActualActual(ActualActual::ISDA);
305
306
307     double tolerance = 1.0e-15;
308
309     // A depo-swap curve
310     std::vector<boost::shared_ptr<RateHelper> > depoSwapInstruments;
311     depoSwapInstruments.push_back(d1w);
312     depoSwapInstruments.push_back(d1m);
313     depoSwapInstruments.push_back(d3m);
314     depoSwapInstruments.push_back(d6m);
315     depoSwapInstruments.push_back(d9m);
316     depoSwapInstruments.push_back(d1y);
317     depoSwapInstruments.push_back(s2y);
318     depoSwapInstruments.push_back(s3y);
319     depoSwapInstruments.push_back(s5y);
320     depoSwapInstruments.push_back(s10y);
321     depoSwapInstruments.push_back(s15y);
322     boost::shared_ptr<YieldTermStructure> depoSwapTermStructure(new
323         PiecewiseFlatForward(settlementDate, depoSwapInstruments,
324             termStructureDayCounter, tolerance));
325
326
327     // A depo-futures-swap curve
328     std::vector<boost::shared_ptr<RateHelper> > depoFutSwapInstruments;
329     depoFutSwapInstruments.push_back(d1w);
330     depoFutSwapInstruments.push_back(d1m);
331     depoFutSwapInstruments.push_back(fut1);
332     depoFutSwapInstruments.push_back(fut2);
333     depoFutSwapInstruments.push_back(fut3);
334     depoFutSwapInstruments.push_back(fut4);
335     depoFutSwapInstruments.push_back(fut5);
336     depoFutSwapInstruments.push_back(fut6);
337     depoFutSwapInstruments.push_back(fut7);
338     depoFutSwapInstruments.push_back(fut8);
339     depoFutSwapInstruments.push_back(s3y);
340     depoFutSwapInstruments.push_back(s5y);
341     depoFutSwapInstruments.push_back(s10y);
342     depoFutSwapInstruments.push_back(s15y);
343     boost::shared_ptr<YieldTermStructure> depoFutSwapTermStructure(new
344         PiecewiseFlatForward(settlementDate, depoFutSwapInstruments,
345             termStructureDayCounter, tolerance));
346
347
348     // A depo-FRA-swap curve
349     std::vector<boost::shared_ptr<RateHelper> > depoFRASwapInstruments;

```

```

350     depoFRASwapInstruments.push_back(d1w);
351     depoFRASwapInstruments.push_back(d1m);
352     depoFRASwapInstruments.push_back(d3m);
353     depoFRASwapInstruments.push_back(fra3x6);
354     depoFRASwapInstruments.push_back(fra6x9);
355     depoFRASwapInstruments.push_back(fra6x12);
356     depoFRASwapInstruments.push_back(s2y);
357     depoFRASwapInstruments.push_back(s3y);
358     depoFRASwapInstruments.push_back(s5y);
359     depoFRASwapInstruments.push_back(s10y);
360     depoFRASwapInstruments.push_back(s15y);
361     boost::shared_ptr<YieldTermStructure> depoFRASwapTermStructure(new
362         PiecewiseFlatForward(settlementDate, depoFRASwapInstruments,
363             termStructureDayCounter, tolerance));
364
365
366     // Term structures that will be used for pricing:
367     // the one used for discounting cash flows
368     Handle<YieldTermStructure> discountingTermStructure;
369     // the one used for forward rate forecasting
370     Handle<YieldTermStructure> forecastingTermStructure;
371
372
373     /*****
374     * SWAPS TO BE PRICED *
375     *****/
376
377     // constant nominal 1,000,000 Euro
378     Real nominal = 1000000.0;
379     // fixed leg
380     Frequency fixedLegFrequency = Annual;
381     BusinessDayConvention fixedLegConvention = Unadjusted;
382     BusinessDayConvention floatingLegConvention = ModifiedFollowing;
383     DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
384     Rate fixedRate = 0.04;
385     DayCounter floatingLegDayCounter = Actual360();
386
387     // floating leg
388     Frequency floatingLegFrequency = Semiannual;
389     boost::shared_ptr<Xibor> euriborIndex(
390         new Euribor6M(forecastingTermStructure));
391     Spread spread = 0.0;
392
393     Integer lenghtInYears = 5;
394     bool payFixedRate = true;
395
396     Date maturity = settlementDate + lenghtInYears*Years;
397     Schedule fixedSchedule(settlementDate, maturity, Period(fixedLegFrequency),
398         calendar, fixedLegConvention, fixedLegConvention,
399         false, false);
400     Schedule floatSchedule(settlementDate, maturity, Period(floatingLegFrequency),
401         calendar, floatingLegConvention, floatingLegConvention,
402         false, false);
403     VanillaSwap spot5YearSwap(
404         payFixedRate, nominal,
405         fixedSchedule, fixedRate, fixedLegDayCounter,
406         floatSchedule, euriborIndex, spread,
407         floatingLegDayCounter, discountingTermStructure);
408
409     Date fwdStart = calendar.advance(settlementDate, 1, Years);
410     Date fwdMaturity = fwdStart + lenghtInYears*Years;
411     Schedule fwdFixedSchedule(fwdStart, fwdMaturity, Period(fixedLegFrequency),
412         calendar, fixedLegConvention, fixedLegConvention,
413         false, false);
414     Schedule fwdFloatSchedule(fwdStart, fwdMaturity, Period(floatingLegFrequency),
415         calendar, floatingLegConvention, floatingLegConvention,
416         false, false);

```

```

417     VanillaSwap oneYearForward5YearSwap(
418         payFixedRate, nominal,
419         fwdFixedSchedule, fixedRate, fixedLegDayCounter,
420         fwdFloatSchedule, euriborIndex, spread,
421         floatingLegDayCounter, discountingTermStructure);
422
423
424     /*****
425     * SWAP PRICING *
426     *****/
427
428     // utilities for reporting
429     std::vector<std::string> headers(4);
430     headers[0] = "term structure";
431     headers[1] = "net present value";
432     headers[2] = "fair spread";
433     headers[3] = "fair fixed rate";
434     std::string separator = " | ";
435     Size width = headers[0].size() + separator.size()
436                 + headers[1].size() + separator.size()
437                 + headers[2].size() + separator.size()
438                 + headers[3].size() + separator.size() - 1;
439     std::string rule(width, '-'), dblrule(width, '=');
440     std::string tab(8, ' ');
441
442     // calculations
443
444     std::cout << dblrule << std::endl;
445     std::cout << "5-year market swap-rate = "
446                 << std::setprecision(2) << io::rate(s5yRate->value())
447                 << std::endl;
448     std::cout << dblrule << std::endl;
449
450     std::cout << tab << "5-years swap paying "
451                 << io::rate(fixedRate) << std::endl;
452     std::cout << headers[0] << separator
453                 << headers[1] << separator
454                 << headers[2] << separator
455                 << headers[3] << separator << std::endl;
456     std::cout << rule << std::endl;
457
458     Real NPV;
459     Rate fairRate;
460     Spread fairSpread;
461
462     // Of course, you're not forced to really use different curves
463     forecastingTermStructure.linkTo(depoSwapTermStructure);
464     discountingTermStructure.linkTo(depoSwapTermStructure);
465
466     NPV = spot5YearSwap.NPV();
467     fairSpread = spot5YearSwap.fairSpread();
468     fairRate = spot5YearSwap.fairRate();
469
470     std::cout << std::setw(headers[0].size())
471                 << "depo-swap" << separator;
472     std::cout << std::setw(headers[1].size())
473                 << std::fixed << std::setprecision(2) << NPV << separator;
474     std::cout << std::setw(headers[2].size())
475                 << io::rate(fairSpread) << separator;
476     std::cout << std::setw(headers[3].size())
477                 << io::rate(fairRate) << separator;
478     std::cout << std::endl;
479
480
481     // let's check that the 5 years swap has been correctly re-priced
482     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
483                 "5-years swap mispriced by ")

```

```

484         << io::rate(std::fabs(fairRate-s5yQuote)));
485
486
487     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
488     discountingTermStructure.linkTo(depoFutSwapTermStructure);
489
490     NPV = spot5YearSwap.NPV();
491     fairSpread = spot5YearSwap.fairSpread();
492     fairRate = spot5YearSwap.fairRate();
493
494     std::cout << std::setw(headers[0].size())
495         << "depo-fut-swap" << separator;
496     std::cout << std::setw(headers[1].size())
497         << std::fixed << std::setprecision(2) << NPV << separator;
498     std::cout << std::setw(headers[2].size())
499         << io::rate(fairSpread) << separator;
500     std::cout << std::setw(headers[3].size())
501         << io::rate(fairRate) << separator;
502     std::cout << std::endl;
503
504     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
505         "5-years swap mispriced!");
506
507
508     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
509     discountingTermStructure.linkTo(depoFRASwapTermStructure);
510
511     NPV = spot5YearSwap.NPV();
512     fairSpread = spot5YearSwap.fairSpread();
513     fairRate = spot5YearSwap.fairRate();
514
515     std::cout << std::setw(headers[0].size())
516         << "depo-FRA-swap" << separator;
517     std::cout << std::setw(headers[1].size())
518         << std::fixed << std::setprecision(2) << NPV << separator;
519     std::cout << std::setw(headers[2].size())
520         << io::rate(fairSpread) << separator;
521     std::cout << std::setw(headers[3].size())
522         << io::rate(fairRate) << separator;
523     std::cout << std::endl;
524
525     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
526         "5-years swap mispriced!");
527
528
529     std::cout << rule << std::endl;
530
531     // now let's price the 1Y forward 5Y swap
532
533     std::cout << tab << "5-years, 1-year forward swap paying "
534         << io::rate(fixedRate) << std::endl;
535     std::cout << headers[0] << separator
536         << headers[1] << separator
537         << headers[2] << separator
538         << headers[3] << separator << std::endl;
539     std::cout << rule << std::endl;
540
541
542     forecastingTermStructure.linkTo(depoSwapTermStructure);
543     discountingTermStructure.linkTo(depoSwapTermStructure);
544
545     NPV = oneYearForward5YearSwap.NPV();
546     fairSpread = oneYearForward5YearSwap.fairSpread();
547     fairRate = oneYearForward5YearSwap.fairRate();
548
549     std::cout << std::setw(headers[0].size())
550         << "depo-swap" << separator;

```

```

551     std::cout << std::setw(headers[1].size())
552               << std::fixed << std::setprecision(2) << NPV << separator;
553     std::cout << std::setw(headers[2].size())
554               << io::rate(fairSpread) << separator;
555     std::cout << std::setw(headers[3].size())
556               << io::rate(fairRate) << separator;
557     std::cout << std::endl;
558
559
560     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
561     discountingTermStructure.linkTo(depoFutSwapTermStructure);
562
563     NPV = oneYearForward5YearSwap.NPV();
564     fairSpread = oneYearForward5YearSwap.fairSpread();
565     fairRate = oneYearForward5YearSwap.fairRate();
566
567     std::cout << std::setw(headers[0].size())
568               << "depo-fut-swap" << separator;
569     std::cout << std::setw(headers[1].size())
570               << std::fixed << std::setprecision(2) << NPV << separator;
571     std::cout << std::setw(headers[2].size())
572               << io::rate(fairSpread) << separator;
573     std::cout << std::setw(headers[3].size())
574               << io::rate(fairRate) << separator;
575     std::cout << std::endl;
576
577
578     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
579     discountingTermStructure.linkTo(depoFRASwapTermStructure);
580
581     NPV = oneYearForward5YearSwap.NPV();
582     fairSpread = oneYearForward5YearSwap.fairSpread();
583     fairRate = oneYearForward5YearSwap.fairRate();
584
585     std::cout << std::setw(headers[0].size())
586               << "depo-FRA-swap" << separator;
587     std::cout << std::setw(headers[1].size())
588               << std::fixed << std::setprecision(2) << NPV << separator;
589     std::cout << std::setw(headers[2].size())
590               << io::rate(fairSpread) << separator;
591     std::cout << std::setw(headers[3].size())
592               << io::rate(fairRate) << separator;
593     std::cout << std::endl;
594
595
596     // now let's say that the 5-years swap rate goes up to 4.60%.
597     // A smarter market element--say, connected to a data source-- would
598     // notice the change itself. Since we're using SimpleQuotes,
599     // we'll have to change the value manually--which forces us to
600     // downcast the handle and use the SimpleQuote
601     // interface. In any case, the point here is that a change in the
602     // value contained in the Quote triggers a new bootstrapping
603     // of the curve and a repricing of the swap.
604
605     boost::shared_ptr<SimpleQuote> fiveYearsRate =
606         boost::dynamic_pointer_cast<SimpleQuote>(s5yRate);
607     fiveYearsRate->setValue(0.0460);
608
609     std::cout << dblrule << std::endl;
610     std::cout << "5-year market swap-rate = "
611               << io::rate(s5yRate->value()) << std::endl;
612     std::cout << dblrule << std::endl;
613
614     std::cout << tab << "5-years swap paying "
615               << io::rate(fixedRate) << std::endl;
616     std::cout << headers[0] << separator
617               << headers[1] << separator

```



```

618             << headers[2] << separator
619             << headers[3] << separator << std::endl;
620         std::cout << rule << std::endl;
621
622         // now get the updated results
623         forecastingTermStructure.linkTo(depoSwapTermStructure);
624         discountingTermStructure.linkTo(depoSwapTermStructure);
625
626         NPV = spot5YearSwap.NPV();
627         fairSpread = spot5YearSwap.fairSpread();
628         fairRate = spot5YearSwap.fairRate();
629
630         std::cout << std::setw(headers[0].size())
631             << "depo-swap" << separator;
632         std::cout << std::setw(headers[1].size())
633             << std::fixed << std::setprecision(2) << NPV << separator;
634         std::cout << std::setw(headers[2].size())
635             << io::rate(fairSpread) << separator;
636         std::cout << std::setw(headers[3].size())
637             << io::rate(fairRate) << separator;
638         std::cout << std::endl;
639
640         QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
641             "5-years swap mispriced!");
642
643
644         forecastingTermStructure.linkTo(depoFutSwapTermStructure);
645         discountingTermStructure.linkTo(depoFutSwapTermStructure);
646
647         NPV = spot5YearSwap.NPV();
648         fairSpread = spot5YearSwap.fairSpread();
649         fairRate = spot5YearSwap.fairRate();
650
651         std::cout << std::setw(headers[0].size())
652             << "depo-fut-swap" << separator;
653         std::cout << std::setw(headers[1].size())
654             << std::fixed << std::setprecision(2) << NPV << separator;
655         std::cout << std::setw(headers[2].size())
656             << io::rate(fairSpread) << separator;
657         std::cout << std::setw(headers[3].size())
658             << io::rate(fairRate) << separator;
659         std::cout << std::endl;
660
661         QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
662             "5-years swap mispriced!");
663
664
665         forecastingTermStructure.linkTo(depoFRASwapTermStructure);
666         discountingTermStructure.linkTo(depoFRASwapTermStructure);
667
668         NPV = spot5YearSwap.NPV();
669         fairSpread = spot5YearSwap.fairSpread();
670         fairRate = spot5YearSwap.fairRate();
671
672         std::cout << std::setw(headers[0].size())
673             << "depo-FRA-swap" << separator;
674         std::cout << std::setw(headers[1].size())
675             << std::fixed << std::setprecision(2) << NPV << separator;
676         std::cout << std::setw(headers[2].size())
677             << io::rate(fairSpread) << separator;
678         std::cout << std::setw(headers[3].size())
679             << io::rate(fairRate) << separator;
680         std::cout << std::endl;
681
682         QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
683             "5-years swap mispriced!");
684

```

```

685         std::cout << rule << std::endl;
686
687         // the 1Y forward 5Y swap changes as well
688
689         std::cout << tab << "5-years, 1-year forward swap paying "
690             << io::rate(fixedRate) << std::endl;
691         std::cout << headers[0] << separator
692             << headers[1] << separator
693             << headers[2] << separator
694             << headers[3] << separator << std::endl;
695         std::cout << rule << std::endl;
696
697
698         forecastingTermStructure.linkTo(depoSwapTermStructure);
699         discountingTermStructure.linkTo(depoSwapTermStructure);
700
701         NPV = oneYearForward5YearSwap.NPV();
702         fairSpread = oneYearForward5YearSwap.fairSpread();
703         fairRate = oneYearForward5YearSwap.fairRate();
704
705         std::cout << std::setw(headers[0].size())
706             << "depo-swap" << separator;
707         std::cout << std::setw(headers[1].size())
708             << std::fixed << std::setprecision(2) << NPV << separator;
709         std::cout << std::setw(headers[2].size())
710             << io::rate(fairSpread) << separator;
711         std::cout << std::setw(headers[3].size())
712             << io::rate(fairRate) << separator;
713         std::cout << std::endl;
714
715
716         forecastingTermStructure.linkTo(depoFutSwapTermStructure);
717         discountingTermStructure.linkTo(depoFutSwapTermStructure);
718
719         NPV = oneYearForward5YearSwap.NPV();
720         fairSpread = oneYearForward5YearSwap.fairSpread();
721         fairRate = oneYearForward5YearSwap.fairRate();
722
723         std::cout << std::setw(headers[0].size())
724             << "depo-fut-swap" << separator;
725         std::cout << std::setw(headers[1].size())
726             << std::fixed << std::setprecision(2) << NPV << separator;
727         std::cout << std::setw(headers[2].size())
728             << io::rate(fairSpread) << separator;
729         std::cout << std::setw(headers[3].size())
730             << io::rate(fairRate) << separator;
731         std::cout << std::endl;
732
733
734         forecastingTermStructure.linkTo(depoFRASwapTermStructure);
735         discountingTermStructure.linkTo(depoFRASwapTermStructure);
736
737         NPV = oneYearForward5YearSwap.NPV();
738         fairSpread = oneYearForward5YearSwap.fairSpread();
739         fairRate = oneYearForward5YearSwap.fairRate();
740
741         std::cout << std::setw(headers[0].size())
742             << "depo-FRA-swap" << separator;
743         std::cout << std::setw(headers[1].size())
744             << std::fixed << std::setprecision(2) << NPV << separator;
745         std::cout << std::setw(headers[2].size())
746             << io::rate(fairSpread) << separator;
747         std::cout << std::setw(headers[3].size())
748             << io::rate(fairRate) << separator;
749         std::cout << std::endl;
750
751         Real seconds = timer.elapsed();

```

```
752     Integer hours = int(seconds/3600);
753     seconds -= hours * 3600;
754     Integer minutes = int(seconds/60);
755     seconds -= minutes * 60;
756     std::cout << " \nRun completed in ";
757     if (hours > 0)
758         std::cout << hours << " h ";
759     if (hours > 0 || minutes > 0)
760         std::cout << minutes << " m ";
761     std::cout << std::fixed << std::setprecision(0)
762         << seconds << " s\n" << std::endl;
763
764     return 0;
765
766 } catch (std::exception& e) {
767     std::cout << e.what() << std::endl;
768     return 1;
769 } catch (...) {
770     std::cout << "unknown error" << std::endl;
771     return 1;
772 }
773 }
774
```

## 9.9 tracing\_example.cpp

This code exemplifies how to insert trace statements to follow the flow of program execution. When compiler under gcc 3.3 and run, the following program will output the following trace:

```

1      trace[1]: Entering int main()
2      trace[2]: Entering int foo(int)
3      trace[3]: Entering int Foo::bar(int)
4      trace[3]: i = 21
5      trace[3]: At line 16 in tracing_example.cpp
6      trace[3]: Wrong answer
7      trace[3]: i = 42
8      trace[3]: Exiting int Foo::bar(int)
9      trace[3]: Entering int Foo::bar(int)
10     trace[3]: i = 42
11     trace[3]: At line 13 in tracing_example.cpp
12     trace[3]: Right answer, but no question
13     trace[3]: i = 42
14     trace[3]: Exiting int Foo::bar(int)
15     trace[2]: Exiting int foo(int)
16     trace[1]: Exiting int main()

```

Of course, a word of warning must be added: adding so much tracing to your code might degrade its readability, at least until we devise an Emacs macro to hide trace statements with a couple of keystrokes.

```

1
2 #include <ql/quantlib.hpp>
3
4 using namespace QuantLib;
5
6 namespace Foo {
7
8     int bar(int i) {
9         QL_TRACE_ENTER_FUNCTION;
10        QL_TRACE_VARIABLE(i);
11
12        if (i == 42) {
13            QL_TRACE_LOCATION;
14            QL_TRACE("Right answer, but no question");
15        } else {
16            QL_TRACE_LOCATION;
17            QL_TRACE("Wrong answer");
18            i *= 2;
19        }
20
21        QL_TRACE_VARIABLE(i);
22        QL_TRACE_EXIT_FUNCTION;
23        return i;
24    }
25
26 }
27
28 int foo(int i) {
29     using namespace Foo;
30     QL_TRACE_ENTER_FUNCTION;
31
32     int j = bar(i);
33     int k = bar(j);
34
35     QL_TRACE_EXIT_FUNCTION;
36     return k;
37 }
38

```

```
39 int main() {  
40  
41     QL_TRACE_ENABLE;  
42  
43     QL_TRACE_ENTER_FUNCTION;  
44  
45     int i = foo(21);  
46  
47     QL_TRACE_EXIT_FUNCTION;  
48     return 0;  
49 }  
50
```



# Chapter 10

## Caveats

**Class `Actual365Fixed`** According to ISDA, "Actual/365" (without "Fixed") is an alias for "Actual/Actual (ISDA)" (see `ActualActual`.) If Actual/365 is not explicitly specified as fixed in an instrument specification, you might want to double-check its meaning.

**Class `BlackSwaptionEngine`** The engine assumes that the exercise date equals the start date of the passed swap.

**Member `BlackVarianceTermStructure::BlackVarianceTermStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Member `BlackVolatilityTermStructure::BlackVolatilityTermStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Member `BlackVolTermStructure::BlackVolTermStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Class `Bond`** Most methods assume that the cashflows are stored sorted by date, the redemption being the last one.

**Member `Bond::cashflows() const`** the returned vector includes the redemption as the last cash flow.

**Member `Bond::cleanPrice() const`** the theoretical price calculated from a flat term structure might differ slightly from the price calculated from the corresponding yield by means of the other overload of this function. If the price from a constant yield is desired, it is advisable to use such other overload.

**Member `Bond::dirtyPrice()` const** the theoretical price calculated from a flat term structure might differ slightly from the price calculated from the corresponding yield by means of the other overload of this function. If the price from a constant yield is desired, it is advisable to use such other overload.

**Class `CADLibor`** This is the rate fixed in London by BBA. Use CDOR if you're interested in the Canadian fixing by IDA.

**Member `Calendar::name()` const** This method is used for output and comparison between calendars. It is **not** meant to be used for writing switch-on-type code.

**Member `CapletVolatilityStructure::CapletVolatilityStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Member `CapVolatilityStructure::CapVolatilityStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Class `Cdor`** This is the rate fixed in Canada by IDA. Use CADLibor if you're interested in the London fixing by BBA.

**Class `CHFLibor`** This is the rate fixed in London by BBA. Use ZIBOR if you're interested in the Zurich fixing.

**Class `CMSCoupon`** 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.

**Class `CompositeInstrument`** Methods that drive the calculation directly (such as `recalculate()`, `freeze()` and others) might not work correctly.

**Class `ConvertibleFixedCouponBond`** Most methods inherited from `Bond` (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

**Class `ConvertibleFloatingRateBond`** Most methods inherited from `Bond` (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.

**Class `ConvertibleZeroCouponBond`** Most methods inherited from `Bond` (such as `yield` or the yield-based `dirtyPrice` and `cleanPrice`) refer to the underlying plain-vanilla bond and do not take convertibility and callability into account.



**Member `Coupon::Coupon`**(Real nominal, const Date &paymentDate, const Date &accrualStartDate, const Date &date) The coupon does not adjust the payment date which must already be a business day.

**Class `CrankNicolson`** The differential operator must be linear for this evolver to work.

**Member `Date::nextIMMdate`**(const Date &d, bool mainCycle=true) The result date is following or equal to the original date.

**Member `Date::IMMcode`**(const Date &date) It raise an exception if the input date is not an IMM date

**Member `DayCounter::name`**() const This method is used for output and comparison between day counters. It is **not** meant to be used for writing switch-on-type code.

**Class `DiscretizedOption`** it is advised that derived classes take care of creating and initializing themselves an instance of the underlying.

**Class `Disposable`** In order to avoid copies in code such as shown above, the conversion from T to Disposable<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

**Class `Euribor`** This is the rate fixed by the ECB. Use EurLibor if you're interested in the London fixing by BBA.

**Class `EURLibor`** This is the rate fixed in London by BBA. Use Euribor if you're interested in the fixing by the ECB.

**Member `ExchangeRateManager::lookup`**(const Currency &source, const Currency &target, Date date=Date(), ExchangeRateManager::ChainType chainType=ChainType::Any) if two or more exchange-rate chains are possible which allow to specify a requested rate, it is unspecified which one is returned.

**Member `FiniteDifferenceModel::rollback`**(array\_type &a, Time from, Time to, Size steps) being this a rollback, from must be a later time than to.

**Member `FiniteDifferenceModel::rollback`**(array\_type &a, Time from, Time to, Size steps, const condition\_type &condition) being this a rollback, from must be a later time than to.

**Class `FixedCouponBondForward`** This class still needs to be rigorously tested

**Class `FixedCouponBondHelper`** This class assumes that the reference date does not change between calls of setTermStructure().

**Class [FloatingRateCoupon](#)** 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.

**Class [Forward](#)** This class still needs to be rigorously tested

**Class [ForwardRateAgreement](#)** This class still needs to be rigorously tested

**Member [ForwardRateStructure::zeroYieldImpl\(Time\) const](#)** This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

**Class [G2SwaptionEngine](#)** The engine assumes that the exercise date equals the start date of the passed swap.

**Member [Handle::Handle\(const boost::shared\\_ptr< T > &h=boostshared\\_ptr< T >\(\), bool registerAsObserver=true\)](#)** see the documentation of the Link class for issues relatives to registerAsObserver.

**Member [Handle::linkTo\(const boost::shared\\_ptr< T > &, bool registerAsObserver=true\)](#)** see the documentation of the Link class for issues relatives to registerAsObserver.

**Class [ImpliedVolTermStructure](#)** 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.

**Class [InArrearIndexedCoupon](#)** 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.

**Class [IncrementalStatistics](#)** high moments are numerically unstable for high average/standard-Deviation ratios.

**Member [Index::name\(\) const=0](#)** This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

**Member [Instrument::setPricingEngine\(const boost::shared\\_ptr< PricingEngine > &\)](#)** calling this method will have no effects in case the **performCalculation** method was overridden in a derived class.

**Member [InterestRate::discountFactor\(Time t\) const](#)** Time must be measured using Interest-Rate's own day counter.

**Member `InterestRate::compoundFactor(Time t) const`** Time must be measured using InterestRate's own day counter.

**Member `InterestRate::equivalentRate(Time t, Compounding comp, Frequency freq=Annual) const`** Time must be measured using the InterestRate's own day counter.

**Member `InterestRate::impliedRate(Real compound, Time t, const DayCounter &resultDC, Compounding comp)`** Time must be measured using the day-counter provided as input.

**Class `JamshidianSwaptionEngine`** The engine assumes that the exercise date equals the start date of the passed swap.

**Class `JPYLibor`** This is the rate fixed in London by BBA. Use TIBOR if you're interested in the Tokio fixing.

**Class `JuQuadraticApproximationEngine`** Barone-Adesi-Whaley critical commodity price calculation is used, it has not been modified to see whether the method of Ju is faster. Ju does not say how he solves the equation for the critical stock price, e.g. Newton method. He just gives the solution. The method of BAW gives answers to the same accuracy as in Ju (1999).

**Member `LazyObject::calculate() const`** 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.

**Member `LazyObject::calculate() const`** Should this method be redefined in derived classes, `LazyObject::calculate()` should be called in the overriding method.

**Class `LiborForwardModelProcess`** this class does not work correctly with Visual C++ 6.

**Member `Link::Link(const boost::shared_ptr< T > &h=boostshared_ptr< T >(), bool registerAsObserver=true)`** see the documentation of the `linkTo()` method for issues relatives to `registerAsObserver`.

**Member `Link::linkTo(const boost::shared_ptr< T > &, bool registerAsObserver=true)`** `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 shared pointer 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 handle to register as observer of such a shared pointer, it is his responsibility to ensure that the handle gets destroyed before the pointed object does.

**Member `LocalVolTermStructure::LocalVolTermStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Class `MCAmericanBasketEngine`** This method is intrinsically weak for out-of-the-money options.

**Class `MCDiscreteAveragingAsianEngine`** control-variate calculation is disabled under VC++6.

**Class `MixedScheme`** The differential operator must be linear for this evolver to work.

**Class `NeumannBC`** 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  $f[1]$ .

**Member `NumericalMethod::partialRollback(DiscretizedAsset &, Time to) const=0`** In version 0.3.7 and earlier, this method was called `rollAlmostBack` method and performed pre-adjustment. This is no longer true; when migrating your code, you'll have to replace calls such as:

```
method->rollAlmostBack(asset,t);
```

with the two statements:

```
method->partialRollback(asset,t);
asset->preAdjustValues();
```

**Member `Observable::operator=(const Observable &)`** notification is sent before the copy constructor has a chance of actually change the data members. Therefore, observers whose `update()` method tries to use their observables will not see the updated values. It is suggested that the `update()` method just raise a flag in order to trigger a later recalculation.

**Member `OneAssetOption::impliedVolatility(Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility)`** currently, this method returns the Black-Scholes implied volatility. It will give inconsistent results if the pricing was performed with any other methods (such as jump-diffusion models.)

**Member `OneAssetOption::impliedVolatility(Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatility)`** options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases the calculation can fail and the result (if any) is almost meaningless. Another possible source of failure is to have a target value that is not attainable with any volatility, e.g., a target value lower than the intrinsic value in the case of American options.

**Class `ParCoupon`** 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.

**Class `PiecewiseYieldCurve`** The bootstrapping algorithm will raise an exception if any two instruments have the same maturity date.

**Class `QuantoEngine`** for the time being, this engine will only work with simple Black-Scholes processes (i.e., no Merton.)

**Class `RandomizedLDS`** Inverting LDS and PRS is possible, but it doesn't make sense.

**Class `RandomSequenceGenerator`** do not use with low-discrepancy sequence generator.

**Member `RateHelper::setTermStructure(YieldTermStructure *)`** Being a pointer and not a shared\_ptr, 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.

**Member `Rounding::Type`** the names of the Floor and Ceiling methods might be misleading. Check the provided reference.

**Member `Settings::evaluationDate()`** a notification is not sent when the evaluation date changes for natural causes—i.e., a date was not explicitly set (which results in today's date being used for pricing) and the current date changes as the clock strikes midnight.

**Class `Short`** 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.

**Class `Short< ParCoupon >`** 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.

**Class `SimpleDayCounter`** 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.

**Member `SingleAssetOption::impliedVolatility(Real targetValue, Real accuracy=1e-4, Size maxEvaluations=100,`**  
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.

**Member `SwapIndex::underlyingSwap(const Date &fixingDate) const`** Relinking the term structure underlying the index will not have effect on the returned swap.

**Class `SwaptionVolatilityCube`** this class is not finalized and its interface might change in subsequent releases.

**Class `SwaptionVolatilityCubeBySabr`** this class is not finalized and its interface might change in subsequent releases.

**Member `SwaptionVolatilityStructure::SwaptionVolatilityStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Member `TermStructure::TermStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Class `Tibor`** This is the rate fixed in Tokio by JBA. Use JPYLibor if you're interested in the London fixing by BBA.

**Class `TreeSwaptionEngine`** This engine is not guaranteed to work if the underlying swap has a start date in the past, i.e., before today's date. When using this engine, prune the initial part of the swap so that it starts at  $t \geq 0$ .

**Class `TridiagonalOperator`** to use real time-dependant algebra, you must overload the corresponding operators in the inheriting time-dependent class.

**Class `TrinomialTree`** The diffusion term of the SDE must be independent of the underlying process.

**Class `UpFrontIndexedCoupon`** 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.

**Class `VarianceSwap`** This class does not manage seasoned variance swaps.

**Member `YieldTermStructure::YieldTermStructure()`** term structures initialized by means of this constructor must manage their own reference date by overriding the `referenceDate()` method.

**Class `Zibor`** This is the rate fixed in Zurich by BBA. Use CHFLibor if you're interested in the London fixing by BBA.

# Chapter 11

## Test Suite

Class **ActualActual** the correctness of the results is checked against known good values.

Class **AnalyticBarrierEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **AnalyticCliquetEngine** • the correctness of the returned value is tested by reproducing results available in literature.  
• the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **AnalyticContinuousFixedLookbackEngine** returned values are verified against results from literature

Class **AnalyticContinuousFloatingLookbackEngine** returned values verified against results from literature

Class **AnalyticContinuousGeometricAveragePriceAsianEngine** • the correctness of the returned value is tested by reproducing results available in literature, and results obtained using a discrete average approximation.  
• the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **AnalyticDigitalAmericanEngine** • the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing results available in literature.  
• the correctness of the returned value in case of asset-or-nothing at-hit digital payoff is tested by reproducing results available in literature.  
• the correctness of the returned value in case of cash-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.  
• the correctness of the returned value in case of asset-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.

- the correctness of the returned greeks in case of cash-or-nothing at-hit digital payoff is tested by reproducing numerical derivatives.

Class **AnalyticDiscreteGeometricAveragePriceAsianEngine** • the correctness of the returned value is tested by reproducing results available in literature.

- the correctness of the available greeks is tested against numerical calculations.

Class **AnalyticDividendEuropeanEngine** the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **AnalyticEuropeanEngine** • the correctness of the returned value is tested by reproducing results available in literature.

- the correctness of the returned greeks is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the implied-volatility calculation is tested by checking that it does not modify the option.
- the correctness of the returned value in case of cash-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of asset-or-nothing digital payoff is tested by reproducing results available in literature.
- the correctness of the returned value in case of gap digital payoff is tested by reproducing results available in literature.
- the correctness of the returned greeks in case of cash-or-nothing digital payoff is tested by reproducing numerical derivatives.

Class **AnalyticHestonEngine** the correctness of the returned value is tested by reproducing results available in web/literature and comparison with Black pricing.

Class **AnalyticPerformanceEngine** the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **Array** construction of arrays is checked in a number of cases

Class **BaroneAdesiWhaleyApproximationEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **BatesEngine** the correctness of the returned value is tested by reproducing results available in web/literature, testing against QuantLib's jump diffusion engine and comparison with Black pricing.



Class **BatesModel** calibration is tested against known values.

Class **BinomialVanillaEngine** the correctness of the returned value is tested by checking it against analytic results.

Class **Bisection** the correctness of the returned values is tested by checking them against known good results.

Class **BivariateCumulativeNormalDistributionDr78** the correctness of the returned value is tested by checking it against known good results.

Class **BivariateCumulativeNormalDistributionWe04DP** the correctness of the returned value is tested by checking it against known good results.

Class **Bjerk SundStenslandApproximationEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **Bond**

- price/yield calculations are cross-checked for consistency.
- price/yield calculations are checked against known good values.

Class **Brazil** the correctness of the returned results is tested against a list of known holidays.

Class **Brent** the correctness of the returned values is tested by checking them against known good results.

Class **Calendar** the methods for adding and removing holidays are tested by inspecting the calendar before and after their invocation.

Class **CapFloor**

- the correctness of the returned value is tested by checking that the price of a cap (resp. floor) decreases (resp. increases) with the strike rate.
- the relationship between the values of caps, floors and the resulting collars is checked.
- the put-call parity between the values of caps, floors and swaps is checked.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the correctness of the returned value is tested by checking it against a known good value.

Class **CompositeQuote** the correctness of the returned values is tested by checking them against numerical calculations.

Class **CompoundForward** • the correctness of the curve is tested by reproducing the input data.

- the correctness of the curve is tested by checking the consistency between returned rates and swaps priced on the curve.

Class **ConvergenceStatistics** results are tested against known good values.

Class **CovarianceDecomposition** cross checked with getCovariance

Class **CubicSpline** the correctness of the returned values is tested by reproducing results available in literature.

Class **CumulativePoissonDistribution** the correctness of the returned value is tested by checking it against known good results.

Class **Date** self-consistency of dates, serial numbers, days of month, months, and weekdays is checked over the whole date range.

Class **DerivedQuote** the correctness of the returned values is tested by checking them against numerical calculations.

Class **DPlusDMinus** the correctness of the returned values is tested by checking them against numerical calculations.

Class **DZero** the correctness of the returned values is tested by checking them against numerical calculations.

Class **ExchangeRate** application of direct and derived exchange rate is tested against calculations.

Class **ExchangeRateManager** lookup of direct, triangulated, and derived exchange rates is tested.

Class **Factorial** the correctness of the returned value is tested by checking it against numerical calculations.

Class **FalsePosition** the correctness of the returned values is tested by checking them against known good results.

---

Class **FaureRsg** the correctness of the returned values is tested by reproducing known good values.

Class **FDEuropeanEngine** the correctness of the returned value is tested by checking it against analytic results.

Class **FixedCouponBond** calculations are tested by checking results against cached values.

Class **FloatingRateBond** calculations are tested by checking results against cached values.

Class **ForwardEngine**

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **ForwardPerformanceEngine**

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **ForwardSpreadedTermStructure**

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Class **GammaFunction** the correctness of the returned value is tested by checking it against known good results.

Class **GaussianQuadrature** the correctness of the result is tested by checking it against known good values.

Class **GenericSequenceStatistics** the correctness of the returned values is tested by checking them against numerical calculations.

Class **Germany** the correctness of the returned results is tested against a list of known holidays.

Class **HaltonRsg**

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Class **HestonModel** calibration is tested against known good values.

Class **HullWhite** calibration results are tested against cached values

Class **ImpliedTermStructure** • the correctness of the returned values is tested by checking them against numerical calculations.  
• observability against changes in the underlying term structure is checked.

Class **InArrearIndexedCoupon** The class is tested by comparing the value of an in-arrear swap against a known good value.

Class **Instrument** observability of class instances is checked.

Class **InterestRate** Converted rates are checked against known good results

Class **InverseCumulativePoisson** the correctness of the returned value is tested by checking it against known good results.

Class **Italy** the correctness of the returned results is tested against a list of known holidays.

Class **JointCalendar** the correctness of the returned results is tested by reproducing the calculations.

Class **JumpDiffusionEngine** • the correctness of the returned value is tested by reproducing results available in literature.  
• the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **JuQuadraticApproximationEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **KronrodIntegral** the correctness of the result is tested by checking it against known good values.

Class **LfmHullWhiteParameterization** the correctness is tested by Monte-Carlo reproduction of caplet & ratchet npvs and comparison with Black pricing.

Class **LiborForwardModel** the correctness is tested using Monte-Carlo Simulation to reproduce swaption npvs, model calibration and exact cap pricing

Class **LiborForwardModelProcess** the correctness is tested by Monte-Carlo reproduction of caplet & ratchet NPVs and comparison with Black pricing.

---

Class **LinearLeastSquaresRegression** the correctness of the returned values is tested by checking their properties.

Class **LongstaffSchwartzPathPricer** the correctness of the returned value is tested by reproducing results available in web/literature

Member **pseudoSqrt**

- the correctness of the results is tested by reproducing known good data.
- the correctness of the results is tested by checking returned values against numerical calculations.

Class **MCAmericanEngine** the correctness of the returned value is tested by reproducing results available in web/literature

Class **MCBarrierEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **MCBasketEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **MCDigitalEngine** the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing known good results.

Class **MCDiscreteArithmeticAPEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **MCDiscreteGeometricAPEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **MCEuropeanEngine** the correctness of the returned value is tested by checking it against analytic results.

Class **MCEuropeanHestonEngine** the correctness of the returned value is tested by reproducing results available in web/literature

Class **MCLongstaffSchwartzEngine** the correctness of the returned value is tested by reproducing results available in web/literature

Class **MCVarianceSwapEngine** returned fair variances checked for consistency with implied volatility curve.

Class **MersenneTwisterUniformRng** the correctness of the returned values is tested by checking them against known good results.

Class **Money** money arithmetic is tested with and without currency conversions.

Class **MultiCubicSpline** interpolated values are checked against the original function.

Class **MultiPathGenerator** the generated paths are checked against cached results

Class **Newton** the correctness of the returned values is tested by checking them against known good results.

Class **NewtonSafe** the correctness of the returned values is tested by checking them against known good results.

Class **NormalDistribution** the correctness of the returned value is tested by checking it against numerical calculations. Cross-checks are also performed against the CumulativeNormalDistribution and InverseCumulativeNormal classes.

Class **OperatorFactory** coefficients are tested against constant BSM operator

Class **PathGenerator** the generated paths are checked against cached results

Class **PiecewiseYieldCurve**

- the correctness of the returned values is tested by checking them against the original inputs.
- the observability of the term structure is tested.

Class **PoissonDistribution** the correctness of the returned value is tested by checking it against known good results.

Class **QuantoEngine**

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **Quote** the observability of class instances is tested.

Class **RandomizedLDS** correct initialization is tested.

---

Class **ReplicatingVarianceSwapEngine** returned fair variances verified against results from literature

Class **Ridder** the correctness of the returned values is tested by checking them against known good results.

Class **Rounding** the correctness of the returned values is tested by checking them against known good results.

Class **Secant** the correctness of the returned values is tested by checking them against known good results.

Class **SeedGenerator** correct initialization of the single instance is tested.

Class **SegmentIntegral** the correctness of the result is tested by checking it against known good values.

Class **SimpleDayCounter** the correctness of the results is checked against known good values.

Class **SimpsonIntegral** the correctness of the result is tested by checking it against known good values.

Class **SobolRsg**

- the correctness of the returned values is tested by reproducing known good values.
- the correctness of the returned values is tested by checking their discrepancy against known good values.

Class **StulzEngine** the correctness of the returned value is tested by reproducing results available in literature.

Class **SVD** the correctness of the returned values is tested by checking their properties.

Class **Swaption**

- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption decreases (resp. increases) with the strike.
- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption increases (resp. decreases) with the spread.
- the correctness of the returned value is tested by checking it against that of a swaption on a swap with no spread and a correspondingly adjusted fixed rate.
- the correctness of the returned value is tested by checking it against a known good value.

- the correctness of the returned value of cash settled swaptions is tested by checking the modified annuity against a value calculated without using the Swaption class.

Class **SymmetricSchurDecomposition** the correctness of the returned values is tested by checking their properties.

Class **TARGET** the correctness of the returned results is tested against a list of known holidays.

Class **TqrEigenDecomposition** the correctness of the result is tested by checking it against known good values.

Class **TrapezoidIntegral** the correctness of the result is tested by checking it against known good values.

Class **TreeSwaptionEngine** calculations are checked against cached results

Class **TreeVanillaSwapEngine** calculations are checked against known good results

Class **UnitedKingdom** the correctness of the returned results is tested against a list of known holidays.

Class **UnitedStates** the correctness of the returned results is tested against a list of known holidays.

Class **VanillaSwap**

- the correctness of the returned value is tested by checking that the price of a swap paying the fair fixed rate is null.
- the correctness of the returned value is tested by checking that the price of a swap receiving the fair floating-rate spread is null.
- the correctness of the returned value is tested by checking that the price of a swap decreases with the paid fixed rate.
- the correctness of the returned value is tested by checking that the price of a swap increases with the received floating-rate spread.
- the correctness of the returned value is tested by checking it against a known good value.

Class **YieldTermStructure** observability against evaluation date changes is checked.

Class **ZeroCouponBond** calculations are tested by checking results against cached values.



---

**Class [ZeroSpreadedTermStructure](#)** • the correctness of the returned values is tested by checking them against numerical calculations.

- observability against changes in the underlying term structure and in the added spread is checked.

**Member [FDAmericanEngine](#)** • the correctness of the returned value is tested by reproducing results available in literature.

- the correctness of the returned greeks is tested by reproducing numerical derivatives.

**Member [FDDividendAmericanEngine](#)** • the correctness of the returned greeks is tested by reproducing numerical derivatives.

- the invariance of the results upon addition of null dividends is tested.

**Member [FDDividendEuropeanEngine](#)** • the correctness of the returned greeks is tested by reproducing numerical derivatives.

- the invariance of the results upon addition of null dividends is tested.

**Member [FDShoutEngine](#)** the correctness of the returned greeks is tested by reproducing numerical derivatives.

**Member [BSMTermOperator](#)** coefficients are tested against constant BSM operator



# Chapter 12

## Todo List

Class [AmericanCondition](#) unify the intrinsicValues/Payoff thing

Class [AmericanExercise](#) check that everywhere the American condition is applied from earliest-Date and not earlier

Class [AmericanPayoffAtExpiry](#) calculate greeks

Class [AmericanPayoffAtHit](#) calculate greeks

Class [AnalyticBarrierEngine](#) rework to avoid repeated casts inside utility methods

Class [AnalyticContinuousGeometricAveragePriceAsianEngine](#) handle seasoned options

Class [AnalyticDigitalAmericanEngine](#) add more greeks (as of now only delta and rho available)

Class [AnalyticDiscreteGeometricAveragePriceAsianEngine](#) implement correct theta, rho, and dividend-rho calculation

Class [BermudanExercise](#) it would be nice to have a way for making a Bermudan with one exercise date equivalent to an European

Class [BicubicSpline](#) revise end conditions

Class [BivariateCumulativeNormalDistributionDr78](#) check accuracy of this algorithm and compare with: 1) Drezner, Z, (1978), Computation of the bivariate normal integral, Mathematics of Computation 32, pp. 277-279. 2) Drezner, Z. and Wesolowsky, G. O. (1990) 'On the Computation of the Bivariate Normal Integral', Journal of Statistical Computation and Simulation 35, pp. 101-107. 3) Drezner, Z (1992) Computation of the Multivariate Normal Integral, ACM Transactions on Mathematics Software 18, pp. 450-460. 4) Drezner, Z (1994) Computation of the Trivariate Normal Integral, Mathematics of Computation 62, pp. 289-294. 5) Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.

Class [BlackVarianceCurve](#) check time extrapolation

Class [BlackVarianceSurface](#) check time extrapolation

Member [BoundaryCondition::Side](#) Generalize for n-dimensional conditions

Class [CapVolatilityVector](#) 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 [Cashflows](#) add tests

Class [Cdor](#) check settlement days and day-count convention.

Class [CliquetOption](#)     • add local/global caps/floors  
                                  • add accrued coupon and last fixing

Class [ContinuousAveragingAsianOption](#) add running average

Class [DirichletBC](#) generalize to time-dependent conditions.

Class [DiscreteGeometricASO](#) add analytical greeks

Member [Event::hasOccurred\(const Date &d, bool includeToday=false\) const](#) make     QL\_-  
                                  TODAYSPAYMENT dynamically configurable?

Class [ExplicitEuler](#) add Richardson extrapolation

Class [FixedCouponBondForward](#) Add preconditions and tests

Class [FixedCouponBondForward](#) Create switch- if coupon goes to seller is toggled on, don't consider income in the  $P_{DirtyFwd}(t)$  calculation.

Class [FixedCouponBondForward](#) Verify this works when the underlying is paper (in which case ignore all AI.)

Class [FloatingRateCoupon](#) add gearing unit test

---

Class **Forward** Add preconditions and tests

Class **ForwardRateAgreement** Add preconditions and tests

Class **ForwardRateAgreement** Should put an instance of ForwardRateAgreement in the FraRateHelper to ensure consistency with the piecewise yield curve.

Class **ForwardRateAgreement** Differentiate between BBA (British)/AFB (French) [assumed here] and ABA (Australian) banker conventions in the calculations.

Class **FuturesRateHelper** implement/refactor constructors with: Index instead of (nMonths, calendar, convention, dayCounter), IMM code

Member **GeneralizedBlackScholesProcess::drift**(Time t, Real x) const revise extrapolation

Member **GeneralizedBlackScholesProcess::diffusion**(Time t, Real x) const revise extrapolation

Class **GenericRiskStatistics** add historical annualized volatility

Member **Index::name**() const=0 add methods returning InterestRate

Class **IntegralEngine** define tolerance for calculate()

Class **InterestRateIndex** add methods returning InterestRate

Class **Jibar** check settlement days and day-count convention.

Class **LogLinearInterpolation** implement primitive, derivative, and secondDerivative functions.

Member **pseudoSqrt** • implement Higham algorithm: Higham "Computing the nearest correlation matrix"

Class **McDiscreteArithmeticASO** continuous-averaging version

Class **MCVarianceSwapEngine** define tolerance of numerical integral and incorporate it in errorEstimate

Class **MixedScheme**     • derive variable theta schemes  
                         • introduce multi time-level schemes.

Class **MultiCubicSpline**     • fix it for Borland compilation  
                         • allow extrapolation as for the other interpolations  
                         • investigate if and how to implement Hyman filters and different boundary conditions

Class **NeumannBC** generalize to time-dependent conditions.

Class **Option::arguments**     • remove `std::vector<Time> stoppingTimes`  
                         • how to handle strike-less option (asian average strike, forward, etc.)?

Class **RandomizedLDS** implement the other randomization algorithms

Member **SampledCurve::valueAtCenter() const** replace or complement with a more general function `valueAt(spot)`

Member **SampledCurve::firstDerivativeAtCenter() const** replace or complement with a more general function `firstDerivativeAt(spot)`

Member **SampledCurve::secondDerivativeAtCenter() const** replace or complement with a more general function `secondDerivativeAt(spot)`

Class **Solver1D**     • clean up the interface so that it is clear whether the accuracy is specified for  $x$  or  $f(x)$ .  
                         • add target value (now the target value is 0.0)

Class **Swaption** add greeks and explicit exercise lag

Class **Tibor** check settlement days.

Class **TimeGrid** 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 **UnitedKingdom** add LIFFE

Class **Xibor** add methods returning `InterestRate`

Class **YieldTermStructure** add derived class `ParSwapTermStructure` similar to `ZeroYieldTermStructure`, `DiscountStructure`, `ForwardRateStructure`

Class **Zibor** check settlement days and day-count.

# Chapter 13

## Known Bugs

**Class [BlackFormula](#)** When the variance is null, division by zero occur during the calculation of delta, delta forward, gamma, gamma forward, rho, dividend rho, vega, and strike sensitivity.

**Class [CompoundForward](#)** swap rates are not reproduced exactly when using indexed coupons. Apparently, some assumption about the swap fixings is hard-coded into the bootstrapping algorithm.

**Class [CoxIngersollRoss](#)** this class was not tested enough to guarantee its functionality.

**Class [ExtendedCoxIngersollRoss](#)** this class was not tested enough to guarantee its functionality.

**Class [G2](#)** This class was not tested enough to guarantee its functionality.

**Class [HullWhite](#)** When the term structure is relinked, the  $r_0$  parameter of the underlying Vasicek model is not updated.

**Class [JuQuadraticApproximationEngine](#)** test fails for Borland compiler

**Class [LocalVolSurface](#)** this class is untested, probably unreliable.

**Class [MultiCubicSpline](#)**

- cannot interpolate at the grid points on the boundary surface of the N-dimensional region
- it does not compile under Borland

**Member [FDDividendAmericanEngine](#)** results are not overly reliable.

**Member [FDDividendAmericanEngine](#)** method `impliedVolatility()` utterly fails

**Member [FDDividendShoutEngine](#)** results are not overly reliable.





## Chapter 14

# Deprecated List

Member **Bond::Bond**(const DayCounter &dayCount, const Calendar &calendar, BusinessDayConvention accrual, const Date &issueDate, const Date &startDate, const Date &endDate, const Date &maturityDate, const Real &faceAmount)  
use constructor with face amount instead

Member **FixedCouponBond::FixedCouponBond**(const Date &issueDate, const Date &startDate, const Date &endDate, const Date &maturityDate, const Real &faceAmount, const Real &couponRate, const DayCounter &dayCount, const Calendar &calendar, BusinessDayConvention accrual)  
use constructor with face amount instead

Member **FloatingRateBond::FloatingRateBond**(const Date &issueDate, const Date &startDate, const Date &endDate, const Date &maturityDate, const Real &faceAmount, const Real &couponRate, const DayCounter &dayCount, const Calendar &calendar, BusinessDayConvention accrual)  
use constructor with face amount instead

Member **Schedule::Schedule**(const Calendar &calendar, const Date &startDate, const Date &endDate, Frequency frequency, const Date &startDate, const Date &endDate, const Date &maturityDate, const Real &faceAmount)  
use other constructors instead

Member **Schedule::Schedule**(const Calendar &calendar, const Date &startDate, const Date &endDate, const Date &maturityDate, const Real &faceAmount, const Real &couponRate, const DayCounter &dayCount, const Calendar &calendar, BusinessDayConvention accrual)  
use other constructors instead

Member **VanillaSwap::VanillaSwap**(bool payFixedRate, Real nominal, const Schedule &fixedSchedule, Rate fixedRate, const Date &startDate, const Date &endDate, const Date &maturityDate, const Real &faceAmount)  
use the other VanillaSwap constructor or Swap instead

Member **Xibor::frequency**() const use tenor() instead

Member **ZeroCouponBond::ZeroCouponBond**(const Date &issueDate, const Date &maturityDate, Integer settlementDays, const Real &faceAmount)  
use constructor with face amount instead

# Index

- ~Error
  - QuantLib::Error, [438](#)
- accruedAmount
  - QuantLib::Bond, [258](#)
- add
  - QuantLib::ExchangeRateManager, [578](#)
  - QuantLib::GeneralStatistics, [685](#)
  - QuantLib::IncrementalStatistics, [737](#)
- addFixing
  - QuantLib::Index, [739](#)
- addFixings
  - QuantLib::Index, [739](#)
- addHoliday
  - QuantLib::Calendar, [278](#)
- addSequence
  - QuantLib::IncrementalStatistics, [737](#)
- adjust
  - QuantLib::Calendar, [278](#)
- adjustValues
  - QuantLib::DiscretizedAsset, [408](#)
- advance
  - QuantLib::Calendar, [278](#)
- amount
  - QuantLib::CashFlow, [305](#)
  - QuantLib::Dividend, [415](#)
  - QuantLib::FixedDividend, [609](#)
  - QuantLib::FixedRateCoupon, [612](#)
  - QuantLib::FloatingRateCoupon, [619](#)
  - QuantLib::FractionalDividend, [647](#)
  - QuantLib::Short, [1029](#)
  - QuantLib::SimpleCashFlow, [1033](#)
- Annual
  - datetime, [109](#)
- apply
  - QuantLib::GeneralizedBlackScholes-  
Process, [682](#)
  - QuantLib::HestonProcess, [709](#)
  - QuantLib::LiborForwardModelProcess,  
[815](#)
  - QuantLib::Merton76Process, [892](#)
  - QuantLib::StochasticProcess, [1067](#)
  - QuantLib::StochasticProcess1D, [1069](#)
  - QuantLib::StochasticProcessArray, [1073](#)
- applyAfterApplying
  - QuantLib::BoundaryCondition, [259](#)
- applyAfterSolving
  - QuantLib::BoundaryCondition, [260](#)
- applyBeforeApplying
  - QuantLib::BoundaryCondition, [259](#)
- applyBeforeSolving
  - QuantLib::BoundaryCondition, [259](#)
- Asian option engines, [115](#)
- AutomatedConversion
  - QuantLib::Money, [898](#)
- averageShortfall
  - QuantLib::GenericRiskStatistics, [694](#)
- BackwardFlatInterpolation
  - QuantLib::BackwardFlatInterpolation, [202](#)
- Barrier option engines, [116](#)
- BaseCurrencyConversion
  - QuantLib::Money, [898](#)
- Basket option engines, [117](#)
- BEJ
  - QuantLib::Indonesia, [744](#)
- BicubicSpline
  - QuantLib::BicubicSpline, [221](#)
- BilinearInterpolation
  - QuantLib::BilinearInterpolation, [223](#)
- Bimonthly
  - datetime, [109](#)
- Biweekly
  - datetime, [109](#)
- blackVarianceImpl
  - QuantLib::BlackVolatilityTermStructure,  
[251](#)
- BlackVarianceTermStructure
  - QuantLib::BlackVarianceTermStructure,  
[248](#)
- BlackVolatilityTermStructure
  - QuantLib::BlackVolatilityTermStructure,  
[250](#)
- blackVolImpl
  - QuantLib::BlackVarianceTermStructure,  
[249](#)
- BlackVolTermStructure
  - QuantLib::BlackVolTermStructure, [253](#)
- BMV
  - QuantLib::Mexico, [894](#)

- Bond
  - QuantLib::Bond, 257
- BoundaryCondition
  - QuantLib::CubicSpline, 374
- bps
  - QuantLib::Cashflows, 308
- BSMTermOperator
  - findiff, 127
- BSSE
  - QuantLib::Slovakia, 1049
- BusinessDayConvention
  - datetime, 108
- businessDaysBetween
  - QuantLib::Calendar, 279
- calculate
  - QuantLib::Instrument, 747
  - QuantLib::LazyObject, 799
- Calendar
  - QuantLib::Calendar, 277
- Calendars, 110
- calibrate
  - QuantLib::CalibratedModel, 284
- Cap/floor engines, 118
- CapletVolatilityStructure
  - QuantLib::CapletVolatilityStructure, 300
- CapVolatilityStructure
  - QuantLib::CapVolatilityStructure, 302
- cashflows
  - QuantLib::Bond, 257
- Ceiling
  - QuantLib::Rounding, 1011
- cleanPrice
  - QuantLib::Bond, 257
- Cliquet option engines, 119
- Closest
  - QuantLib::Rounding, 1011
- compoundFactor
  - QuantLib::InterestRate, 751
- compoundForwardImpl
  - QuantLib::ExtendedDiscountCurve, 588
- computePlain
  - QuantLib::DriftCalculator, 426
- computeReduced
  - QuantLib::DriftCalculator, 426
- ConversionType
  - QuantLib::Money, 898
- convexity
  - QuantLib::Cashflows, 309
- convexityBias
  - QuantLib::HullWhite, 715
- correlationMatrix
  - QuantLib::CovarianceDecomposition, 365
- Coupon
  - QuantLib::Coupon, 364
- covariance
  - QuantLib::Abcd, 159
  - QuantLib::EulerDiscretization, 442
  - QuantLib::G2ForwardProcess, 655
  - QuantLib::G2Process, 657
  - QuantLib::LiborForwardModelProcess, 815
  - QuantLib::StochasticProcess, 1066
  - QuantLib::StochasticProcessArray, 1073
- CovarianceDecomposition
  - QuantLib::CovarianceDecomposition, 365
- CubicSpline
  - QuantLib::CubicSpline, 374
- Currencies and FX rates, 103
- Currency
  - QuantLib::Currency, 382
- Daily
  - datetime, 109
- Date and time calculations, 107
- datetime
  - Annual, 109
  - Bimonthly, 109
  - Biweekly, 109
  - BusinessDayConvention, 108
  - Daily, 109
  - EveryFourthMonth, 109
  - Following, 108
  - Frequency, 109
  - ModifiedFollowing, 108
  - ModifiedPreceding, 108
  - MonthEndReference, 109
  - Monthly, 109
  - NoFrequency, 109
  - Once, 109
  - Preceding, 108
  - Quarterly, 109
  - Semiannual, 109
  - Unadjusted, 109
  - UnadjustedMonthEnd, 109
  - Weekday, 109
  - Weekly, 109
- Day counters, 113
- DayCounter
  - QuantLib::DayCounter, 393
- Debugging macros, 154
- debugMacros
  - QL\_TRACE, 155
  - QL\_TRACE\_DISABLE, 154
  - QL\_TRACE\_ENABLE, 154
  - QL\_TRACE\_ENTER\_FUNCTION, 155
  - QL\_TRACE\_EXIT\_FUNCTION, 155
  - QL\_TRACE\_LOCATION, 156

- QL\_TRACE\_ON, 155
- QL\_TRACE\_VARIABLE, 156
- DEFINE\_SEQUENCE\_STAT\_CONST\_-  
METHOD\_DOUBLE  
sequencestatistics.hpp, 1432
- DEFINE\_SEQUENCE\_STAT\_CONST\_-  
METHOD\_VOID  
sequencestatistics.hpp, 1431
- Derived
  - QuantLib::ExchangeRate, 576
- Design patterns, 141
- diffusion
  - QuantLib::EulerDiscretization, 441
  - QuantLib::GeneralizedBlackScholes-  
Process, 682
- Direct
  - QuantLib::ExchangeRate, 576
- dirtyPrice
  - QuantLib::Bond, 257, 258
- discount
  - QuantLib::YieldTermStructure, 1181
- DiscountCurve
  - yieldtermstructures, 145
- discountFactor
  - QuantLib::InterestRate, 751
- discountImpl
  - QuantLib::CompoundForward, 334
  - QuantLib::ForwardRateStructure, 639
  - QuantLib::ZeroYieldStructure, 1189
- Down
  - QuantLib::Rounding, 1011
- downsideDeviation
  - QuantLib::GenericRiskStatistics, 693
  - QuantLib::IncrementalStatistics, 737
- downsideVariance
  - QuantLib::GenericRiskStatistics, 693
  - QuantLib::IncrementalStatistics, 737
- drift
  - QuantLib::EulerDiscretization, 441
  - QuantLib::GeneralizedBlackScholes-  
Process, 682
- DriftCalculator
  - QuantLib::DriftCalculator, 426
- duration
  - QuantLib::Cashflows, 308
- earliestDate
  - QuantLib::RateHelper, 1003
- equivalentRate
  - QuantLib::InterestRate, 752
- Error
  - QuantLib::Error, 438
- errorEstimate
  - QuantLib::GeneralStatistics, 684
  - QuantLib::IncrementalStatistics, 736
- errors.hpp
  - QL\_ASSERT, 1273
  - QL\_ENSURE, 1273
  - QL\_FAIL, 1273
  - QL\_REQUIRE, 1273
- Eurex
  - QuantLib::Germany, 700
- evaluationDate
  - QuantLib::Settings, 1025
- EveryFourthMonth
  - datetime, 109
- evolve
  - QuantLib::LiborForwardModelProcess,  
815
  - QuantLib::StochasticProcess, 1066
  - QuantLib::StochasticProcess1D, 1069
- Exchange
  - QuantLib::Italy, 777
  - QuantLib::UnitedKingdom, 1150
- ExchangeRate
  - QuantLib::ExchangeRate, 577
- expectation
  - QuantLib::G2ForwardProcess, 655
  - QuantLib::G2Process, 657
  - QuantLib::HullWhiteForwardProcess, 719
  - QuantLib::HullWhiteProcess, 721
  - QuantLib::OrnsteinUhlenbeckProcess, 960
  - QuantLib::StochasticProcess, 1066
  - QuantLib::StochasticProcess1D, 1069
  - QuantLib::StochasticProcessArray, 1073
- expectationValue
  - QuantLib::GeneralStatistics, 685
- expectedShortfall
  - QuantLib::GenericRiskStatistics, 693
- FDAmericanEngine
  - vanillaengines, 124
- FDDividendAmericanEngine
  - vanillaengines, 124
- FDDividendEuropeanEngine
  - vanillaengines, 125
- FDDividendShoutEngine
  - vanillaengines, 125
- FDShoutEngine
  - vanillaengines, 125
- fetchResults
  - QuantLib::AssetSwap, 193
  - QuantLib::ForwardVanillaOption, 645
  - QuantLib::Instrument, 747
  - QuantLib::MultiAssetOption, 906
  - QuantLib::OneAssetOption, 945
  - QuantLib::OneAssetStrikedOption, 949
  - QuantLib::QuantoVanillaOption, 997

- QuantLib::VanillaSwap, [1163](#)
- QuantLib::VarianceSwap, [1167](#)
- Financial instruments, [131](#)
- findiff
  - BSMTermOperator, [127](#)
- Finite-differences framework, [126](#)
- FirstDerivative
  - QuantLib::CubicSpline, [374](#)
- firstDerivativeAtCenter
  - QuantLib::SampledCurve, [1016](#)
- FixedCouponBond
  - QuantLib::FixedCouponBond, [603](#)
- FixedCouponBondForward
  - QuantLib::FixedCouponBondForward, [605](#)
- fixing
  - QuantLib::Index, [739](#)
  - QuantLib::InterestRateIndex, [754](#)
- FloatingRateBond
  - QuantLib::FloatingRateBond, [616](#)
- Floor
  - QuantLib::Rounding, [1011](#)
- Following
  - datetime, [108](#)
- format
  - QuantLib::Currency, [382](#)
- Forward option engines, [120](#)
- ForwardFlatInterpolation
  - QuantLib::ForwardFlatInterpolation, [628](#)
- forwardRate
  - QuantLib::YieldTermStructure, [1181](#)
- forwardValue
  - QuantLib::Forward, [625](#)
- FrankfurtStockExchange
  - QuantLib::Germany, [700](#)
- freeze
  - QuantLib::LazyObject, [799](#)
- Frequency
  - datetime, [109](#)
- frequency
  - QuantLib::Xibor, [1178](#)
- gaussianDownsideDeviation
  - QuantLib::GenericGaussianStatistics, [689](#)
- gaussianDownsideVariance
  - QuantLib::GenericGaussianStatistics, [688](#)
- gaussianExpectedShortfall
  - QuantLib::GenericGaussianStatistics, [689](#)
- gaussianPercentile
  - QuantLib::GenericGaussianStatistics, [689](#)
- gaussianPotentialUpside
  - QuantLib::GenericGaussianStatistics, [689](#)
- gaussianRegret
  - QuantLib::GenericGaussianStatistics, [689](#)
- gaussianTopPercentile
  - QuantLib::GenericGaussianStatistics, [689](#)
- gaussianValueAtRisk
  - QuantLib::GenericGaussianStatistics, [689](#)
- Generic macros, [149](#)
- GovernmentBond
  - QuantLib::UnitedStates, [1153](#)
- Handle
  - QuantLib::Handle, [705](#)
- hasOccurred
  - QuantLib::Event, [573](#)
- HKEx
  - QuantLib::HongKong, [712](#)
- ICEX
  - QuantLib::Iceland, [724](#)
- IMMcode
  - QuantLib::Date, [390](#)
- IMMdate
  - QuantLib::Date, [390](#)
- impliedRate
  - QuantLib::InterestRate, [751](#), [752](#)
- impliedVolatility
  - QuantLib::OneAssetOption, [944](#)
  - QuantLib::SingleAssetOption, [1042](#)
- impliedYield
  - QuantLib::Forward, [625](#)
- incomeDiscountCurve\_
  - QuantLib::Forward, [625](#)
- instantaneousCovariance
  - QuantLib::Abcd, [158](#)
- instantaneousVariance
  - QuantLib::Abcd, [158](#)
- instantaneousVolatility
  - QuantLib::Abcd, [158](#)
- irr
  - QuantLib::Cashflows, [308](#)
- isBusinessDay
  - QuantLib::Calendar, [277](#)
- isEndOfMonth
  - QuantLib::Calendar, [278](#)
- isExpired
  - QuantLib::ForwardRateAgreement, [635](#)
- isHoliday
  - QuantLib::Calendar, [277](#)
- isOnTime
  - QuantLib::DiscretizedAsset, [408](#)
- isWeekend
  - QuantLib::Calendar, [278](#)
- Iterator support, [152](#)
- iteratorMacros
  - QL\_FULL\_ITERATOR\_SUPPORT, [152](#)
- itmAssetProbability

- QuantLib::BlackFormula, 236
- itmCashProbability
  - QuantLib::BlackFormula, 236
- KnuthUniformRng
  - QuantLib::KnuthUniformRng, 789
- KRX
  - QuantLib::SouthKorea, 1058
- kurtosis
  - QuantLib::GeneralStatistics, 684
  - QuantLib::IncrementalStatistics, 736
- Lagrange
  - QuantLib::CubicSpline, 374
- latestDate
  - QuantLib::FixedCouponBondHelper, 608
  - QuantLib::RateHelper, 1003
- Lattice methods, 135
- LecuyerUniformRng
  - QuantLib::LecuyerUniformRng, 802
- LevenbergMarquardt
  - QuantLib::LevenbergMarquardt, 804
- limitMacros
  - QL\_EPSILON, 150
  - QL\_MAX\_INTEGER, 150
  - QL\_MAX\_REAL, 150
  - QL\_MIN\_INTEGER, 150
  - QL\_MIN\_POSITIVE\_REAL, 150
  - QL\_MIN\_REAL, 150
- LinearInterpolation
  - QuantLib::LinearInterpolation, 817
- Link
  - QuantLib::Link, 822
- linkTo
  - QuantLib::Handle, 705
  - QuantLib::Link, 822
- localVolImpl
  - QuantLib::LocalVolCurve, 834
- LocalVolTermStructure
  - QuantLib::LocalVolTermStructure, 838
- LogLinearInterpolation
  - QuantLib::LogLinearInterpolation, 840
- lookup
  - QuantLib::ExchangeRateManager, 578
- mandatoryTimes
  - QuantLib::DiscretizedAsset, 408
  - QuantLib::DiscretizedDiscountBond, 410
  - QuantLib::DiscretizedOption, 411
- Market
  - QuantLib::Argentina, 184
  - QuantLib::Brazil, 264
  - QuantLib::CzechRepublic, 386
  - QuantLib::Germany, 700
  - QuantLib::HongKong, 712
  - QuantLib::Iceland, 724
  - QuantLib::India, 742
  - QuantLib::Indonesia, 744
  - QuantLib::Italy, 777
  - QuantLib::Mexico, 894
  - QuantLib::Singapore, 1040
  - QuantLib::Slovakia, 1049
  - QuantLib::SouthKorea, 1058
  - QuantLib::Taiwan, 1107
  - QuantLib::Ukraine, 1148
  - QuantLib::UnitedKingdom, 1150
  - QuantLib::UnitedStates, 1153
- Math tools, 138
- max
  - QuantLib::GeneralStatistics, 685
  - QuantLib::IncrementalStatistics, 737
- mean
  - QuantLib::GeneralStatistics, 684
  - QuantLib::IncrementalStatistics, 736
- MersenneTwisterUniformRng
  - QuantLib::MersenneTwisterUniformRng, 890
- Merval
  - QuantLib::Argentina, 184
- min
  - QuantLib::GeneralStatistics, 684
  - QuantLib::IncrementalStatistics, 737
- miscMacros
  - QL\_DUMMY\_RETURN, 149
  - QL\_IO\_INIT, 149
- ModifiedFollowing
  - datetime, 108
- ModifiedPreceding
  - datetime, 108
- MonotonicCubicSpline
  - QuantLib::MonotonicCubicSpline, 899
- Monte Carlo framework, 140
- MonthEndReference
  - datetime, 109
- Monthly
  - datetime, 109
- name
  - QuantLib::Calendar, 277
  - QuantLib::DayCounter, 393
  - QuantLib::Index, 738
  - QuantLib::InterestRateIndex, 754
- NaturalCubicSpline
  - QuantLib::NaturalCubicSpline, 917
- NaturalMonotonicCubicSpline
  - QuantLib::NaturalMonotonicCubicSpline, 918
- NERC

- QuantLib::UnitedStates, 1153
- next
  - QuantLib::KnuthUniformRng, 789
  - QuantLib::LecuyerUniformRng, 802
  - QuantLib::MersenneTwisterUniformRng, 890
- nextIMMdate
  - QuantLib::Date, 390
- nextRandomizer
  - QuantLib::RandomizedLDS, 1000
- nextWeekday
  - QuantLib::Date, 390
- NoConversion
  - QuantLib::Money, 898
- NoFrequency
  - datetime, 109
- None
  - QuantLib::Rounding, 1011
- NotAKnot
  - QuantLib::CubicSpline, 374
- notifyObservers
  - QuantLib::Observable, 939
- npv
  - QuantLib::Cashflows, 307, 308
- NSE
  - QuantLib::India, 742
- nthWeekday
  - QuantLib::Date, 390
- Numeric limits, 150
- Numeric types, 101
- NYSE
  - QuantLib::UnitedStates, 1153
- Once
  - datetime, 109
- operator()
  - QuantLib::EndCriteria, 435
- operator+=
  - QuantLib::Matrix, 859
- operator=
  - QuantLib::Observable, 939
- operator==
  - QuantLib::Calendar, 279
  - QuantLib::DayCounter, 393
- Output manipulators, 153
- parRate
  - QuantLib::YieldTermStructure, 1181
- partialRollback
  - QuantLib::Lattice, 794
  - QuantLib::NumericalMethod, 934
  - QuantLib::TsiveriotisFernandesLattice, 1138
- percentile
  - QuantLib::GeneralStatistics, 685
- performCalculations
  - QuantLib::Bond, 258
  - QuantLib::CompositeInstrument, 331
  - QuantLib::FixedCouponBondForward, 606
  - QuantLib::Forward, 625
  - QuantLib::ForwardRateAgreement, 636
  - QuantLib::Instrument, 748
  - QuantLib::LazyObject, 799
  - QuantLib::Stock, 1074
  - QuantLib::Swap, 1080
  - QuantLib::VarianceSwap, 1167
- Periodic
  - QuantLib::CubicSpline, 374
- postAdjustValues
  - QuantLib::DiscretizedAsset, 408
- postAdjustValuesImpl
  - QuantLib::DiscretizedAsset, 409
  - QuantLib::DiscretizedOption, 412
- potentialUpside
  - QuantLib::GenericRiskStatistics, 693
- preAdjustValues
  - QuantLib::DiscretizedAsset, 408
- preAdjustValuesImpl
  - QuantLib::DiscretizedAsset, 409
- Preceding
  - datetime, 108
- Pricing engines, 114
- PSE
  - QuantLib::CzechRepublic, 386
- pseudoSqrt
  - QuantLib::Matrix, 859
- ql/argsandresults.hpp, 1191
- ql/calendar.hpp, 1193
- ql/Calendars/argentina.hpp, 1195
- ql/Calendars/australia.hpp, 1196
- ql/Calendars/brazil.hpp, 1197
- ql/Calendars/canada.hpp, 1198
- ql/Calendars/china.hpp, 1199
- ql/Calendars/czechrepublic.hpp, 1200
- ql/Calendars/denmark.hpp, 1201
- ql/Calendars/finland.hpp, 1202
- ql/Calendars/germany.hpp, 1203
- ql/Calendars/hongkong.hpp, 1204
- ql/Calendars/hungary.hpp, 1205
- ql/Calendars/iceland.hpp, 1206
- ql/Calendars/india.hpp, 1207
- ql/Calendars/indonesia.hpp, 1208
- ql/Calendars/italy.hpp, 1209
- ql/Calendars/japan.hpp, 1210
- ql/Calendars/jointcalendar.hpp, 1211
- ql/Calendars/mexico.hpp, 1212



- ql/Calendars/newzealand.hpp, 1213
- ql/Calendars/norway.hpp, 1214
- ql/Calendars>nullcalendar.hpp, 1215
- ql/Calendars/poland.hpp, 1216
- ql/Calendars/saudiArabia.hpp, 1217
- ql/Calendars/singapore.hpp, 1218
- ql/Calendars/slovakia.hpp, 1219
- ql/Calendars/southAfrica.hpp, 1220
- ql/Calendars/southKorea.hpp, 1221
- ql/Calendars/sweden.hpp, 1222
- ql/Calendars/switzerland.hpp, 1223
- ql/Calendars/taiwan.hpp, 1224
- ql/Calendars/target.hpp, 1225
- ql/Calendars/turkey.hpp, 1226
- ql/Calendars/ukraine.hpp, 1227
- ql/Calendars/unitedKingdom.hpp, 1228
- ql/Calendars/unitedStates.hpp, 1229
- ql/capvolstructures.hpp, 1230
- ql/cashflow.hpp, 1231
- ql/CashFlows/analysis.hpp, 1232
- ql/CashFlows/cashflowvectors.hpp, 1233
- ql/CashFlows/cmscoupon.hpp, 1234
- ql/CashFlows/conundrumpricer.hpp, 1236
- ql/CashFlows/coupon.hpp, 1237
- ql/CashFlows/dividend.hpp, 1238
- ql/CashFlows/fixedratecoupon.hpp, 1239
- ql/CashFlows/floatingratecoupon.hpp, 1240
- ql/CashFlows/inarrearindexedcoupon.hpp, 1241
- ql/CashFlows/indexedcashflowvectors.hpp, 1242
- ql/CashFlows/indexedcoupon.hpp, 1243
- ql/CashFlows/parcoupon.hpp, 1244
- ql/CashFlows/shortfloatingcoupon.hpp, 1245
- ql/CashFlows/shortindexedcoupon.hpp, 1246
- ql/CashFlows/simplecashflow.hpp, 1247
- ql/CashFlows/timebasket.hpp, 1248
- ql/CashFlows/upfrontindexedcoupon.hpp, 1249
- ql/Currencies/africa.hpp, 1250
- ql/Currencies/america.hpp, 1251
- ql/Currencies/asia.hpp, 1252
- ql/Currencies/europe.hpp, 1254
- ql/Currencies/exchangeratemanager.hpp, 1257
- ql/Currencies/oceania.hpp, 1258
- ql/currency.hpp, 1259
- ql/date.hpp, 1260
- ql/daycounter.hpp, 1263
- ql/DayCounters/actual360.hpp, 1264
- ql/DayCounters/actual365fixed.hpp, 1265
- ql/DayCounters/actualactual.hpp, 1266
- ql/DayCounters/business252.hpp, 1267
- ql/DayCounters/one.hpp, 1268
- ql/DayCounters/simpledaycounter.hpp, 1269
- ql/DayCounters/thirty360.hpp, 1270
- ql/discretizedasset.hpp, 1271
- ql/errors.hpp, 1272
- ql/event.hpp, 1275
- ql/exchangerate.hpp, 1276
- ql/exercise.hpp, 1277
- ql/FiniteDifferences/americancondition.hpp, 1278
- ql/FiniteDifferences/boundarycondition.hpp, 1279
- ql/FiniteDifferences/bsmoperator.hpp, 1280
- ql/FiniteDifferences/bsmtermoperator.hpp, 1281
- ql/FiniteDifferences/cranknicolson.hpp, 1282
- ql/FiniteDifferences/dminus.hpp, 1283
- ql/FiniteDifferences/dplus.hpp, 1284
- ql/FiniteDifferences/dplusdminus.hpp, 1285
- ql/FiniteDifferences/dzero.hpp, 1286
- ql/FiniteDifferences/expliciteuler.hpp, 1287
- ql/FiniteDifferences/fdtypedefs.hpp, 1288
- ql/FiniteDifferences/finitedifferencemodel.hpp, 1289
- ql/FiniteDifferences/impliciteuler.hpp, 1290
- ql/FiniteDifferences/mixedscheme.hpp, 1291
- ql/FiniteDifferences/onefactoroperator.hpp, 1292
- ql/FiniteDifferences/operatorfactory.hpp, 1293
- ql/FiniteDifferences/operatortraits.hpp, 1294
- ql/FiniteDifferences/paralelolver.hpp, 1295
- ql/FiniteDifferences/pde.hpp, 1296
- ql/FiniteDifferences/pdebsm.hpp, 1297
- ql/FiniteDifferences/pdeshortrate.hpp, 1298
- ql/FiniteDifferences/shoutcondition.hpp, 1299
- ql/FiniteDifferences/stepcondition.hpp, 1300
- ql/FiniteDifferences/tridiagonaloperator.hpp, 1301
- ql/FiniteDifferences/zerocondition.hpp, 1302
- ql/grid.hpp, 1303
- ql/handle.hpp, 1304
- ql/index.hpp, 1305
- ql/Indexes/audlibor.hpp, 1306
- ql/Indexes/cadlibor.hpp, 1307
- ql/Indexes/cdor.hpp, 1308
- ql/Indexes/chflibor.hpp, 1309
- ql/Indexes/dkklbor.hpp, 1310
- ql/Indexes/euribor.hpp, 1311
- ql/Indexes/euriborswapfixa.hpp, 1314
- ql/Indexes/euriborswapfixifr.hpp, 1316
- ql/Indexes/eurlibor.hpp, 1318
- ql/Indexes/eurliborswapfixa.hpp, 1320
- ql/Indexes/eurliborswapfixb.hpp, 1322
- ql/Indexes/eurliborswapfixifr.hpp, 1324
- ql/Indexes/gbplibor.hpp, 1326
- ql/Indexes/indexmanager.hpp, 1327



- ql/Indexes/interestrategyindex.hpp, 1328
- ql/Indexes/jlibor.hpp, 1329
- ql/Indexes/jpylibor.hpp, 1330
- ql/Indexes/libor.hpp, 1331
- ql/Indexes/nzdlibor.hpp, 1332
- ql/Indexes/swapindex.hpp, 1333
- ql/Indexes/tibor.hpp, 1334
- ql/Indexes/trlibor.hpp, 1335
- ql/Indexes/usdlibor.hpp, 1336
- ql/Indexes/xibor.hpp, 1337
- ql/Indexes/zibor.hpp, 1338
- ql/instrument.hpp, 1339
- ql/Instruments/asianooption.hpp, 1340
- ql/Instruments/assetswap.hpp, 1341
- ql/Instruments/barrieroption.hpp, 1342
- ql/Instruments/basketoption.hpp, 1343
- ql/Instruments/bond.hpp, 1344
- ql/Instruments/callabilityschedule.hpp, 1345
- ql/Instruments/capfloor.hpp, 1346
- ql/Instruments/cliquestoption.hpp, 1348
- ql/Instruments/compositeinstrument.hpp, 1349
- ql/Instruments/convertiblebond.hpp, 1350
- ql/Instruments/dividendschedule.hpp, 1352
- ql/Instruments/dividendvanillaoption.hpp, 1353
- ql/Instruments/europeanoption.hpp, 1354
- ql/Instruments/fixedcouponbond.hpp, 1355
- ql/Instruments/fixedcouponbondforward.hpp, 1356
- ql/Instruments/floatingratebond.hpp, 1357
- ql/Instruments/forward.hpp, 1358
- ql/Instruments/forwardrateagreement.hpp, 1359
- ql/Instruments/forwardvanillaoption.hpp, 1360
- ql/Instruments/lookbackoption.hpp, 1361
- ql/Instruments/multiassetoption.hpp, 1362
- ql/Instruments/oneassetoption.hpp, 1363
- ql/Instruments/oneassetstrikedoption.hpp, 1364
- ql/Instruments/payoffs.hpp, 1365
- ql/Instruments/quantoforwardvanillaoption.hpp, 1366
- ql/Instruments/quantovanillaoption.hpp, 1367
- ql/Instruments/stock.hpp, 1368
- ql/Instruments/swap.hpp, 1369
- ql/Instruments/swapoption.hpp, 1370
- ql/Instruments/vanillaoption.hpp, 1371
- ql/Instruments/vanillaswap.hpp, 1372
- ql/Instruments/varianceswap.hpp, 1373
- ql/Instruments/zerocouponbond.hpp, 1374
- ql/interestrate.hpp, 1375
- ql/Lattices/binomialtree.hpp, 1376
- ql/Lattices/bsmlattice.hpp, 1378
- ql/Lattices/lattice.hpp, 1379
- ql/Lattices/lattice1d.hpp, 1380
- ql/Lattices/lattice2d.hpp, 1381
- ql/Lattices/tflattice.hpp, 1382
- ql/Lattices/tree.hpp, 1383
- ql/Lattices/trinomialtree.hpp, 1384
- ql/MarketModels/driftcalculator.hpp, 1385
- ql/Math/array.hpp, 1386
- ql/Math/backwardflatinterpolation.hpp, 1387
- ql/Math/beta.hpp, 1388
- ql/Math/bicubicsplineinterpolation.hpp, 1389
- ql/Math/bilinearinterpolation.hpp, 1390
- ql/Math/binomialdistribution.hpp, 1391
- ql/Math/bivariate normal distribution.hpp, 1392
- ql/Math/chisquaredistribution.hpp, 1393
- ql/Math/choleskydecomposition.hpp, 1394
- ql/Math/comparison.hpp, 1395
- ql/Math/convergence statistics.hpp, 1396
- ql/Math/cubicspline.hpp, 1397
- ql/Math/discrepancy statistics.hpp, 1398
- ql/Math/errorfunction.hpp, 1399
- ql/Math/extrapolation.hpp, 1400
- ql/Math/factorial.hpp, 1401
- ql/Math/forwardflatinterpolation.hpp, 1402
- ql/Math/functional.hpp, 1403
- ql/Math/gammadistribution.hpp, 1404
- ql/Math/gaussianorthogonalpolynomial.hpp, 1405
- ql/Math/gaussianquadratures.hpp, 1406
- ql/Math/gaussianstatistics.hpp, 1408
- ql/Math/generalstatistics.hpp, 1409
- ql/Math/incompletegamma.hpp, 1410
- ql/Math/incrementalstatistics.hpp, 1411
- ql/Math/interpolation.hpp, 1412
- ql/Math/interpolation2D.hpp, 1413
- ql/Math/kronrodintegral.hpp, 1414
- ql/Math/lexicographicalview.hpp, 1415
- ql/Math/linearinterpolation.hpp, 1416
- ql/Math/linearleast squares regression.hpp, 1417
- ql/Math/loglinearinterpolation.hpp, 1418
- ql/Math/matrix.hpp, 1419
- ql/Math/multicubicspline.hpp, 1420
- ql/Math/normaldistribution.hpp, 1422
- ql/Math/poissondistribution.hpp, 1423
- ql/Math/primenumbers.hpp, 1424
- ql/Math/pseudosqrt.hpp, 1425
- ql/Math/riskstatistics.hpp, 1426
- ql/Math/rounding.hpp, 1427
- ql/Math/sabrinterpolation.hpp, 1428
- ql/Math/sampledcurve.hpp, 1429
- ql/Math/segmentintegral.hpp, 1430
- ql/Math/sequence statistics.hpp, 1431
- ql/Math/simpsonintegral.hpp, 1433
- ql/Math/statistics.hpp, 1434

- ql/Math/svd.hpp, 1435
- ql/Math/symmetricschurdecomposition.hpp, 1436
- ql/Math/tqreigendecomposition.hpp, 1437
- ql/Math/transformedgrid.hpp, 1438
- ql/Math/trapezoidintegral.hpp, 1439
- ql/money.hpp, 1440
- ql/MonteCarlo/brownianbridge.hpp, 1441
- ql/MonteCarlo/earlyexercisepathpricer.hpp, 1442
- ql/MonteCarlo/getcovariance.hpp, 1443
- ql/MonteCarlo/longstaffschwartzpathpricer.hpp, 1444
- ql/MonteCarlo/lsmbasissystem.hpp, 1445
- ql/MonteCarlo/mctraits.hpp, 1446
- ql/MonteCarlo/mctypedefs.hpp, 1447
- ql/MonteCarlo/montecarlomodel.hpp, 1448
- ql/MonteCarlo/multipath.hpp, 1449
- ql/MonteCarlo/multipathgenerator.hpp, 1450
- ql/MonteCarlo/path.hpp, 1451
- ql/MonteCarlo/pathgenerator.hpp, 1452
- ql/MonteCarlo/pathpricer.hpp, 1453
- ql/MonteCarlo/sample.hpp, 1454
- ql/numericalmethod.hpp, 1455
- ql/Optimization/armijo.hpp, 1456
- ql/Optimization/conjugategradient.hpp, 1457
- ql/Optimization/constraint.hpp, 1458
- ql/Optimization/costfunction.hpp, 1459
- ql/Optimization/criteria.hpp, 1460
- ql/Optimization/leastsquare.hpp, 1461
- ql/Optimization/levenbergmarquardt.hpp, 1462
- ql/Optimization/linesearch.hpp, 1463
- ql/Optimization/lmdif.hpp, 1464
- ql/Optimization/method.hpp, 1465
- ql/Optimization/problem.hpp, 1466
- ql/Optimization/simplex.hpp, 1467
- ql/Optimization/steepestdescent.hpp, 1468
- ql/option.hpp, 1469
- ql/Patterns/bridge.hpp, 1470
- ql/Patterns/composite.hpp, 1471
- ql/Patterns/curiouslyrecurring.hpp, 1472
- ql/Patterns/lazyobject.hpp, 1473
- ql/Patterns/observable.hpp, 1474
- ql/Patterns/singleton.hpp, 1475
- ql/Patterns/visitor.hpp, 1476
- ql/payoff.hpp, 1477
- ql/period.hpp, 1478
- ql/position.hpp, 1480
- ql/Pricers/discretegeometricaso.hpp, 1481
- ql/Pricers/mccliquetoption.hpp, 1482
- ql/Pricers/mcdiscretearithmeticsaso.hpp, 1483
- ql/Pricers/mceverest.hpp, 1484
- ql/Pricers/mchimalaya.hpp, 1485
- ql/Pricers/mcmaxbasket.hpp, 1486
- ql/Pricers/mcpagoda.hpp, 1487
- ql/Pricers/mcperformanceoption.hpp, 1488
- ql/Pricers/mcpricer.hpp, 1489
- ql/Pricers/singleassetoption.hpp, 1490
- ql/prices.hpp, 1491
- ql/pricingengine.hpp, 1492
- ql/PricingEngines/americanpayofffatexpiry.hpp, 1493
- ql/PricingEngines/americanpayoffathit.hpp, 1494
- ql/PricingEngines/Asian/analytic\_cont\_geom\_av\_price.hpp, 1495
- ql/PricingEngines/Asian/analytic\_discr\_geom\_av\_price.hpp, 1496
- ql/PricingEngines/Asian/mc\_discr\_arith\_av\_price.hpp, 1497
- ql/PricingEngines/Asian/mc\_discr\_geom\_av\_price.hpp, 1498
- ql/PricingEngines/Asian/mcdiscreteasianengine.hpp, 1499
- ql/PricingEngines/Barrier/analyticbarrierengine.hpp, 1500
- ql/PricingEngines/Barrier/mcbarrierengine.hpp, 1501
- ql/PricingEngines/Basket/mcamericanbasketengine.hpp, 1502
- ql/PricingEngines/Basket/mcbasketengine.hpp, 1503
- ql/PricingEngines/Basket/stulzengine.hpp, 1504
- ql/PricingEngines/blackformula.hpp, 1505
- ql/PricingEngines/blackmodel.hpp, 1506
- ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp, 1507
- ql/PricingEngines/CapFloor/blackcapfloorengine.hpp, 1508
- ql/PricingEngines/CapFloor/discretizedcapfloor.hpp, 1509
- ql/PricingEngines/CapFloor/mchullwhiteengine.hpp, 1510
- ql/PricingEngines/CapFloor/treecapfloorengine.hpp, 1511
- ql/PricingEngines/Cliquet/analyticcliquetengine.hpp, 1512
- ql/PricingEngines/Cliquet/analyticperformanceengine.hpp, 1513
- ql/PricingEngines/Forward/forwardengine.hpp, 1514
- ql/PricingEngines/Forward/forwardperformanceengine.hpp, 1515
- ql/PricingEngines/Forward/mcvarianceswapengine.hpp, 1516

- [ql/PricingEngines/Forward/replicatingvarianceswapengine.hpp](#), 1517  
[ql/PricingEngines/genericmodelengine.hpp](#), 1518  
[ql/PricingEngines/greeks.hpp](#), 1519  
[ql/PricingEngines/Hybrid/binomialconvertibleengine.hpp](#), 1520  
[ql/PricingEngines/Hybrid/discretizedconvertible.hpp](#), 1521  
[ql/PricingEngines/latticeshortratemodelengine.hpp](#), 1522  
[ql/PricingEngines/Lookback/analyticcontinuousfixedlookback.hpp](#), 1523  
[ql/PricingEngines/Lookback/analyticcontinuousfloatinglookback.hpp](#), 1524  
[ql/PricingEngines/mclongstaffschwartzengine.hpp](#), 1525  
[ql/PricingEngines/mcsimulation.hpp](#), 1526  
[ql/PricingEngines/Quanto/quantoengine.hpp](#), 1527  
[ql/PricingEngines/Swaption/blackswaptionengine.hpp](#), 1528  
[ql/PricingEngines/Swaption/discretizedswaption.hpp](#), 1529  
[ql/PricingEngines/Swaption/g2swaptionengine.hpp](#), 1530  
[ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp](#), 1531  
[ql/PricingEngines/Swaption/lfmswaptionengine.hpp](#), 1532  
[ql/PricingEngines/Swaption/treeswaptionengine.hpp](#), 1533  
[ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp](#), 1534  
[ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp](#), 1535  
[ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp](#), 1536  
[ql/PricingEngines/Vanilla/analytichestonengine.hpp](#), 1537  
[ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp](#), 1538  
[ql/PricingEngines/Vanilla/batesengine.hpp](#), 1539  
[ql/PricingEngines/Vanilla/binomialengine.hpp](#), 1540  
[ql/PricingEngines/Vanilla/bjerkstundstenslandengine.hpp](#), 1541  
[ql/PricingEngines/Vanilla/discretizedvanillaoptionengine.hpp](#), 1542  
[ql/PricingEngines/Vanilla/fdamericanengine.hpp](#), 1543  
[ql/PricingEngines/Vanilla/fdbermudanengine.hpp](#), 1544  
[ql/PricingEngines/Vanilla/fdconditions.hpp](#), 1545  
[ql/PricingEngines/Vanilla/fddividendamericanengine.hpp](#), 1546  
[ql/PricingEngines/Vanilla/fddividendengine.hpp](#), 1547  
[ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp](#), 1548  
[ql/PricingEngines/Vanilla/fddividendshoutengine.hpp](#), 1549  
[ql/PricingEngines/Vanilla/fdeuropeanengine.hpp](#), 1550  
[ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp](#), 1551  
[ql/PricingEngines/Vanilla/fdshoutengine.hpp](#), 1552  
[ql/PricingEngines/Vanilla/fdstepconditionengine.hpp](#), 1553  
[ql/PricingEngines/Vanilla/fdvanillaengine.hpp](#), 1554  
[ql/PricingEngines/Vanilla/integralengine.hpp](#), 1555  
[ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp](#), 1556  
[ql/PricingEngines/Vanilla/juquadraticengine.hpp](#), 1557  
[ql/PricingEngines/Vanilla/mcamericanengine.hpp](#), 1558  
[ql/PricingEngines/Vanilla/mcdigitalengine.hpp](#), 1559  
[ql/PricingEngines/Vanilla/mceuropeanengine.hpp](#), 1560  
[ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp](#), 1561  
[ql/PricingEngines/Vanilla/mcvanillaengine.hpp](#), 1562  
[ql/Processes/blackscholesprocess.hpp](#), 1563  
[ql/Processes/eulordiscretization.hpp](#), 1564  
[ql/Processes/forwardmeasureprocess.hpp](#), 1565  
[ql/Processes/g2process.hpp](#), 1566  
[ql/Processes/geometricbrownianprocess.hpp](#), 1567  
[ql/Processes/hestonprocess.hpp](#), 1568  
[ql/Processes/hullwhiteprocess.hpp](#), 1569  
[ql/Processes/lfmcovarParams.hpp](#), 1570  
[ql/Processes/lfmhullwhiteparam.hpp](#), 1571  
[ql/Processes/lfmprocess.hpp](#), 1572  
[ql/Processes/merton76process.hpp](#), 1573  
[ql/Processes/ornsteinuhlenbeckprocess.hpp](#), 1574  
[ql/Processes/squarerootprocess.hpp](#), 1575  
[ql/Processes/stochasticprocessarray.hpp](#), 1576  
[ql/qldefines.hpp](#), 1577  
[ql/quote.hpp](#), 1579

- [ql/RandomNumbers/boxmullergaussianrng.hpp](#), [ql/ShortRateModels/model.hpp](#), 1612  
[1580](#) [ql/ShortRateModels/onefactormodel.hpp](#), 1613  
[ql/RandomNumbers/centrallimitgaussianrng.hpp](#), [ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp](#),  
[1581](#) [1614](#)  
[ql/RandomNumbers/faurersg.hpp](#), 1582 [ql/ShortRateModels/OneFactorModels/coxingersollross.hpp](#),  
[ql/RandomNumbers/haltonrsg.hpp](#), 1583 [1615](#)  
[ql/RandomNumbers/inversecumulativerng.hpp](#), [ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp](#),  
[1584](#) [1616](#)  
[ql/RandomNumbers/inversecumulativersg.hpp](#), [ql/ShortRateModels/OneFactorModels/hullwhite.hpp](#),  
[1585](#) [1617](#)  
[ql/RandomNumbers/knuthuniformrng.hpp](#), [ql/ShortRateModels/OneFactorModels/vasicek.hpp](#),  
[1586](#) [1618](#)  
[ql/RandomNumbers/lecuyeruniformrng.hpp](#), [ql/ShortRateModels/parameter.hpp](#), 1619  
[1587](#) [ql/ShortRateModels/twofactormodel.hpp](#), 1620  
[ql/RandomNumbers/mt19937uniformrng.hpp](#), [ql/ShortRateModels/TwoFactorModels/batesmodel.hpp](#),  
[1588](#) [1621](#)  
[ql/RandomNumbers/randomizedlds.hpp](#), 1589 [ql/ShortRateModels/TwoFactorModels/g2.hpp](#),  
[ql/RandomNumbers/randomsequencegenerator.hpp](#), [1622](#)  
[1590](#) [ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp](#),  
[ql/RandomNumbers/rngtraits.hpp](#), 1591 [1623](#)  
[ql/RandomNumbers/seedgenerator.hpp](#), 1593 [ql/solver1d.hpp](#), 1624  
[ql/RandomNumbers/sobolrsg.hpp](#), 1594 [ql/Solvers1D/bisection.hpp](#), 1625  
[ql/schedule.hpp](#), 1595 [ql/Solvers1D/brent.hpp](#), 1626  
[ql/settings.hpp](#), 1596 [ql/Solvers1D/falseposition.hpp](#), 1627  
[ql/ShortRateModels/calibrationhelper.hpp](#), [ql/Solvers1D/newton.hpp](#), 1628  
[1597](#) [ql/Solvers1D/newtonsafe.hpp](#), 1629  
[ql/ShortRateModels/CalibrationHelpers/caphelper.hpp](#), [ql/Solvers1D/ridder.hpp](#), 1630  
[1598](#) [ql/Solvers1D/secant.hpp](#), 1631  
[ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp](#), [ql/ShortRateModels/process.hpp](#), 1632  
[1599](#) [ql/swaptionvolstructure.hpp](#), 1633  
[ql/ShortRateModels/CalibrationHelpers/swaptionmodelhelper.hpp](#), [ql/termstructure.hpp](#), 1634  
[1600](#) [ql/TermStructures/bondhelpers.hpp](#), 1635  
[ql/ShortRateModels/LiborMarketModels/lfmcovariancestructure.hpp](#), [ql/TermStructures/bootstraptraits.hpp](#), 1636  
[1601](#) [ql/TermStructures/compoundforward.hpp](#),  
[ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp](#), 1637  
[1602](#) [ql/TermStructures/discountcurve.hpp](#), 1638  
[ql/ShortRateModels/LiborMarketModels/lmconstantdrifttermstructure.hpp](#), [ql/TermStructures/flatforward.hpp](#), 1641  
[1603](#) [ql/TermStructures/forwardcurve.hpp](#), 1642  
[ql/ShortRateModels/LiborMarketModels/lmconstantwrappermodel.hpp](#), [ql/TermStructures/forwardspreadedtermstructure.hpp](#),  
[1604](#) [1643](#)  
[ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp](#), [ql/TermStructures/forwardstructure.hpp](#), 1644  
[1605](#) [ql/TermStructures/impliedtermstructure.hpp](#),  
[ql/ShortRateModels/LiborMarketModels/lmextlinearmodel.hpp](#), [1645](#)  
[1607](#) [ql/TermStructures/piecewiseflatforward.hpp](#),  
[ql/ShortRateModels/LiborMarketModels/lmfixedvoltermstructure.hpp](#), [1646](#)  
[1608](#) [ql/TermStructures/piecewiseyieldcurve.hpp](#),  
[ql/ShortRateModels/LiborMarketModels/lmlineartermstructure.hpp](#), [1647](#)  
[1609](#) [ql/TermStructures/piecewisezerospreadedtermstructure.hpp](#),  
[ql/ShortRateModels/LiborMarketModels/lmlinearvoltermstructure.hpp](#), [1648](#)  
[1610](#) [ql/TermStructures/quantotermstructure.hpp](#),  
[ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp](#), [1649](#)  
[1611](#)

- ql/TermStructures/ratehelpers.hpp, 1650
- ql/TermStructures/zerocurve.hpp, 1651
- ql/TermStructures/zerospreadedtermstructure.hpp, 1652
- ql/TermStructures/zeroyieldstructure.hpp, 1653
- ql/timegrid.hpp, 1654
- ql/timeseries.hpp, 1655
- ql/types.hpp, 1656
- ql/Utilities/clone.hpp, 1658
- ql/Utilities/dataformatters.hpp, 1659
- ql/Utilities/dataparsers.hpp, 1660
- ql/Utilities/disposable.hpp, 1661
- ql/Utilities/null.hpp, 1662
- ql/Utilities/observablevalue.hpp, 1663
- ql/Utilities/steppingiterator.hpp, 1664
- ql/Utilities/strings.hpp, 1665
- ql/Utilities/tracing.hpp, 1666
- ql/Volatilities/blackconstantvol.hpp, 1668
- ql/Volatilities/blackvariancecurve.hpp, 1669
- ql/Volatilities/blackvariancesurface.hpp, 1670
- ql/Volatilities/capflatvolvector.hpp, 1671
- ql/Volatilities/capletconstantvol.hpp, 1672
- ql/Volatilities/capletvariancecurve.hpp, 1673
- ql/Volatilities/cmsmarket.hpp, 1674
- ql/Volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp, 1675
- ql/Volatilities/localconstantvol.hpp, 1676
- ql/Volatilities/localvolcurve.hpp, 1677
- ql/Volatilities/localvolsurface.hpp, 1678
- ql/Volatilities/smilesection.hpp, 1679
- ql/Volatilities/swaptionconstantvol.hpp, 1680
- ql/Volatilities/swaptionvolcube.hpp, 1681
- ql/Volatilities/swaptionvolcubebySabr.hpp, 1682
- ql/Volatilities/swaptionvolmatrix.hpp, 1683
- ql/volatilitymodel.hpp, 1684
- ql/VolatilityModels/constantestimator.hpp, 1685
- ql/VolatilityModels/garch.hpp, 1686
- ql/VolatilityModels/garmanklass.hpp, 1687
- ql/VolatilityModels/simplelocalestimator.hpp, 1688
- ql/voltermstructure.hpp, 1689
- ql/yieldtermstructure.hpp, 1690
- QL\_ASSERT
  - errors.hpp, 1273
- QL\_DUMMY\_RETURN
  - miscMacros, 149
- QL\_ENSURE
  - errors.hpp, 1273
- QL\_EPSILON
  - limitMacros, 150
- QL\_FAIL
  - errors.hpp, 1273
- QL\_FULL\_ITERATOR\_SUPPORT
  - iteratorMacros, 152
- QL\_IO\_INIT
  - miscMacros, 149
- QL\_MAX\_INTEGER
  - limitMacros, 150
- QL\_MAX\_REAL
  - limitMacros, 150
- QL\_MIN\_INTEGER
  - limitMacros, 150
- QL\_MIN\_POSITIVE\_REAL
  - limitMacros, 150
- QL\_MIN\_REAL
  - limitMacros, 150
- QL\_REQUIRE
  - errors.hpp, 1273
- QL\_TRACE
  - debugMacros, 155
- QL\_TRACE\_DISABLE
  - debugMacros, 154
- QL\_TRACE\_ENABLE
  - debugMacros, 154
- QL\_TRACE\_ENTER\_FUNCTION
  - debugMacros, 155
- QL\_TRACE\_EXIT\_FUNCTION
  - debugMacros, 155
- QL\_TRACE\_LOCATION
  - debugMacros, 156
- QL\_TRACE\_ON
  - debugMacros, 155
- QL\_TRACE\_VARIABLE
  - debugMacros, 156
- QL\_TYPENAME
  - templateMacros, 151
- QuantLib macros, 148
- QuantLib::Abcd, 157
- QuantLib::Abcd
  - covariance, 159
  - instantaneousCovariance, 158
  - instantaneousVariance, 158
  - instantaneousVolatility, 158
  - variance, 159
  - volatility, 158
- QuantLib::AbcdSquared, 160
- QuantLib::Actual360, 161
- QuantLib::Actual365Fixed, 162
- QuantLib::ActualActual, 163
- QuantLib::AcyclicVisitor, 164
- QuantLib::AdditiveEQPBinoMialTree, 165
- QuantLib::AffineModel, 166
- QuantLib::AmericanCondition, 167
- QuantLib::AmericanExercise, 168
- QuantLib::AmericanPayoffAtExpiry, 169



- QuantLib::AmericanPayoffAtHit, [170](#)
- QuantLib::AnalyticBarrierEngine, [171](#)
- QuantLib::AnalyticCapFloorEngine, [172](#)
- QuantLib::AnalyticCliquetEngine, [173](#)
- QuantLib::AnalyticContinuousFixedLookbackEngine, [174](#)
- QuantLib::AnalyticContinuousFloatingLookbackEngine, [175](#)
- QuantLib::AnalyticContinuousGeometricAveragePriceAsianEngine, [176](#)
- QuantLib::AnalyticDigitalAmericanEngine, [177](#)
- QuantLib::AnalyticDiscreteGeometricAveragePriceAsianEngine, [178](#)
- QuantLib::AnalyticDividendEuropeanEngine, [179](#)
- QuantLib::AnalyticEuropeanEngine, [180](#)
- QuantLib::AnalyticHestonEngine, [181](#)
- QuantLib::AnalyticPerformanceEngine, [182](#)
- QuantLib::Argentina, [183](#)
  - Merval, [184](#)
- QuantLib::Argentina
  - Market, [184](#)
- QuantLib::Arguments, [185](#)
- QuantLib::ArmijoLineSearch, [186](#)
- QuantLib::Array, [187](#)
- QuantLib::ARSCurrency, [190](#)
- QuantLib::AssetOrNothingPayoff, [191](#)
- QuantLib::AssetSwap, [192](#)
- QuantLib::AssetSwap
  - fetchResults, [193](#)
  - setupArguments, [193](#)
- QuantLib::AssetSwap::arguments, [194](#)
- QuantLib::AssetSwap::results, [195](#)
- QuantLib::ATSCurrency, [196](#)
- QuantLib::AUDCurrency, [197](#)
- QuantLib::AUDLibor, [198](#)
- QuantLib::Australia, [199](#)
- QuantLib::Average, [200](#)
- QuantLib::BackwardFlat, [201](#)
- QuantLib::BackwardFlatInterpolation, [202](#)
- QuantLib::BackwardFlatInterpolation
  - BackwardFlatInterpolation, [202](#)
- QuantLib::BaroneAdesiWhaleyApproximationEngine, [203](#)
- QuantLib::Barrier, [204](#)
- QuantLib::BarrierOption, [205](#)
- QuantLib::BarrierOption
  - setupArguments, [206](#)
- QuantLib::BarrierOption::arguments, [207](#)
- QuantLib::BarrierOption::engine, [208](#)
- QuantLib::BasketOption, [209](#)
- QuantLib::BasketOption
  - setupArguments, [210](#)
- QuantLib::BasketOption::arguments, [211](#)
- QuantLib::BasketOption::engine, [212](#)
- QuantLib::BatesEngine, [213](#)
- QuantLib::BatesModel, [215](#)
- QuantLib::BDTCurrency, [216](#)
- QuantLib::BEFCurrency, [217](#)
- QuantLib::BermudanExercise, [218](#)
- QuantLib::BGLCurrency, [219](#)
- QuantLib::BicubicSpline, [220](#)
- QuantLib::BicubicSpline
  - BicubicSpline, [221](#)
- QuantLib::BilinearInterpolation, [222](#)
- QuantLib::BilinearInterpolation
  - BilinearInterpolation, [223](#)
- QuantLib::BinomialConvertibleEngine, [224](#)
- QuantLib::BinomialDistribution, [225](#)
- QuantLib::BinomialTree, [226](#)
- QuantLib::BinomialVanillaEngine, [227](#)
- QuantLib::Bisection, [228](#)
- QuantLib::BivariateCumulativeNormalDistributionDr78, [229](#)
- QuantLib::BivariateCumulativeNormalDistributionWe04DP, [230](#)
- QuantLib::Bjerk SundStenslandApproximationEngine, [231](#)
- QuantLib::BlackCapFloorEngine, [232](#)
- QuantLib::BlackCapFloorEngine
  - update, [232](#)
- QuantLib::BlackConstantVol, [233](#)
- QuantLib::BlackFormula, [235](#)
- QuantLib::BlackFormula
  - itmAssetProbability, [236](#)
  - itmCashProbability, [236](#)
- QuantLib::BlackKarasinski, [237](#)
- QuantLib::BlackKarasinski::Dynamics, [238](#)
- QuantLib::BlackProcess, [239](#)
- QuantLib::BlackScholesLattice, [240](#)
- QuantLib::BlackScholesMertonProcess, [241](#)
- QuantLib::BlackScholesProcess, [242](#)
- QuantLib::BlackSwaptionEngine, [243](#)
- QuantLib::BlackSwaptionEngine
  - update, [243](#)
- QuantLib::BlackVarianceCurve, [244](#)
- QuantLib::BlackVarianceSurface, [246](#)
- QuantLib::BlackVarianceTermStructure, [248](#)
- QuantLib::BlackVarianceTermStructure
  - BlackVarianceTermStructure, [248](#)
  - blackVolImpl, [249](#)
- QuantLib::BlackVolatilityTermStructure, [250](#)
- QuantLib::BlackVolatilityTermStructure
  - blackVarianceImpl, [251](#)
  - BlackVolatilityTermStructure, [250](#)

- QuantLib::BlackVolTermStructure, 252
- QuantLib::BlackVolTermStructure
  - BlackVolTermStructure, 253
- QuantLib::Bond, 255
- QuantLib::Bond
  - accruedAmount, 258
  - Bond, 257
  - cashflows, 257
  - cleanPrice, 257
  - dirtyPrice, 257, 258
  - performCalculations, 258
  - yield, 257, 258
- QuantLib::BoundaryCondition, 259
- QuantLib::BoundaryCondition
  - applyAfterApplying, 259
  - applyAfterSolving, 260
  - applyBeforeApplying, 259
  - applyBeforeSolving, 259
  - setTime, 260
  - Side, 259
- QuantLib::BoundaryConstraint, 261
- QuantLib::BoxMullerGaussianRng, 262
- QuantLib::Brazil, 263
  - Settlement, 264
- QuantLib::Brazil
  - Market, 264
- QuantLib::Brent, 265
- QuantLib::Bridge, 266
- QuantLib::BRLCurrency, 267
- QuantLib::BrownianBridge, 268
- QuantLib::BSMOperator, 269
- QuantLib::Business252, 270
- QuantLib::BYRCurrency, 271
- QuantLib::CADCurrency, 272
- QuantLib::CADLibor, 273
- QuantLib::Calendar, 274
- QuantLib::Calendar
  - addHoliday, 278
  - adjust, 278
  - advance, 278
  - businessDaysBetween, 279
  - Calendar, 277
  - isBusinessDay, 277
  - isEndOfMonth, 278
  - isHoliday, 277
  - isWeekend, 278
  - name, 277
  - operator==, 279
  - removeHoliday, 278
- QuantLib::Calendar::OrthodoxImpl, 280
- QuantLib::Calendar::WesternImpl, 281
- QuantLib::CalendarImpl, 282
- QuantLib::CalibratedModel, 283
- QuantLib::CalibratedModel
  - calibrate, 284
  - update, 284
- QuantLib::CalibrationHelper, 285
- QuantLib::CalibrationHelper
  - update, 286
- QuantLib::Callability, 287
- QuantLib::Callability::Price, 288
- QuantLib::Canada, 289
- QuantLib::Cap, 290
- QuantLib::CapFloor, 291
- QuantLib::CapFloor
  - setupArguments, 292
- QuantLib::CapFloor::arguments, 293
- QuantLib::CapFloor::engine, 294
- QuantLib::CapFloor::results, 295
- QuantLib::CapHelper, 296
- QuantLib::CapletConstantVolatility, 297
- QuantLib::CapletVolatilityStructure, 299
- QuantLib::CapletVolatilityStructure
  - CapletVolatilityStructure, 300
- QuantLib::CapVolatilityStructure, 301
- QuantLib::CapVolatilityStructure
  - CapVolatilityStructure, 302
- QuantLib::CapVolatilityVector, 303
- QuantLib::CapVolatilityVector
  - update, 304
- QuantLib::CashFlow, 305
- QuantLib::CashFlow
  - amount, 305
- QuantLib::Cashflows, 307
- QuantLib::Cashflows
  - bps, 308
  - convexity, 309
  - duration, 308
  - irr, 308
  - npv, 307, 308
- QuantLib::CashOrNothingPayoff, 310
- QuantLib::Cdor, 311
- QuantLib::CeilingTruncation, 312
- QuantLib::CHFCurrency, 313
- QuantLib::CHFLibor, 314
- QuantLib::China, 315
- QuantLib::CLGaussianRng, 316
- QuantLib::CliquetOption, 317
- QuantLib::CliquetOption
  - setupArguments, 318
- QuantLib::CliquetOption::arguments, 319
- QuantLib::CliquetOption::engine, 320
- QuantLib::Clone, 321
- QuantLib::ClosestRounding, 322
- QuantLib::CLPCurrency, 323
- QuantLib::CMSCoupon, 324
- QuantLib::CNYPurrency, 326
- QuantLib::Collar, 327

- QuantLib::Composite, 328
- QuantLib::CompositeConstraint, 329
- QuantLib::CompositeInstrument, 330
- QuantLib::CompositeInstrument
  - performCalculations, 331
- QuantLib::CompositeQuote, 332
- QuantLib::CompositeQuote
  - update, 332
- QuantLib::CompoundForward, 333
- QuantLib::CompoundForward
  - discountImpl, 334
  - zeroYieldImpl, 334
- QuantLib::ConjugateGradient, 335
- QuantLib::ConstantEstimator, 336
- QuantLib::ConstantParameter, 337
- QuantLib::Constraint, 338
- QuantLib::ConstraintImpl, 339
- QuantLib::ContinuousAveragingAsianOption, 340
- QuantLib::ContinuousAveragingAsianOption
  - setupArguments, 341
- QuantLib::ContinuousAveragingAsianOption::arguments, 342
- QuantLib::ContinuousAveragingAsianOption::engine, 343
- QuantLib::ContinuousFixedLookbackOption, 344
- QuantLib::ContinuousFixedLookbackOption
  - setupArguments, 345
- QuantLib::ContinuousFixedLookbackOption::arguments, 346
- QuantLib::ContinuousFixedLookbackOption::engine, 347
- QuantLib::ContinuousFloatingLookbackOption, 348
- QuantLib::ContinuousFloatingLookback-Option
  - setupArguments, 349
- QuantLib::ContinuousFloatingLookbackOption::arguments, 350
- QuantLib::ContinuousFloatingLookbackOption::engine, 351
- QuantLib::ConundrumPricer, 352
- QuantLib::ConundrumPricerByNumericalIntegration, 354
- QuantLib::ConvergenceStatistics, 355
- QuantLib::ConvertibleBond::option::arguments, 356
- QuantLib::ConvertibleBond::option::engine, 357
- QuantLib::ConvertibleFixedCouponBond, 358
- QuantLib::ConvertibleFloatingRateBond, 359
- QuantLib::ConvertibleZeroCouponBond, 360
- QuantLib::COPCurrency, 361
- QuantLib::CostFunction, 362
- QuantLib::Coupon, 363
- QuantLib::Coupon
  - Coupon, 364
- QuantLib::CovarianceDecomposition, 365
- QuantLib::CovarianceDecomposition
  - correlationMatrix, 365
  - CovarianceDecomposition, 365
  - standardDeviations, 365
  - variances, 365
- QuantLib::CoxIngersollRoss, 366
- QuantLib::CoxIngersollRoss::Dynamics, 368
- QuantLib::CoxRossRubinstein, 369
- QuantLib::CrankNicolson, 370
- QuantLib::Cubic, 372
- QuantLib::CubicSpline, 373
  - FirstDerivative, 374
  - Lagrange, 374
  - NotAKnot, 374
  - Periodic, 374
  - SecondDerivative, 374
- QuantLib::CubicSpline
  - BoundaryCondition, 374
- QuantLib::CumulativeBinomialDistribution, 375
- QuantLib::CumulativeNormalDistribution, 376
- QuantLib::CumulativePoissonDistribution, 377
- QuantLib::CuriouslyRecurringTemplate, 378
- QuantLib::Currency, 379
- QuantLib::Currency
  - Currency, 382
  - format, 382
- QuantLib::CurveState, 383
- QuantLib::CYPCurrency, 384
- QuantLib::CzechRepublic, 385
- QuantLib::CzechRepublic
  - Market, 386
- QuantLib::CZKCurrency, 387
- QuantLib::Date, 388
- QuantLib::Date
  - IMMcode, 390
  - IMMdate, 390
  - nextIMMdate, 390
  - nextWeekday, 390
  - nthWeekday, 390
- QuantLib::DayCounter, 392
- QuantLib::DayCounter
  - DayCounter, 393
  - name, 393
  - operator==, 393



- QuantLib::DayCounterImpl, 394
- QuantLib::DEMCurrency, 395
- QuantLib::Denmark, 396
- QuantLib::DepositRateHelper, 397
- QuantLib::DerivedQuote, 398
- QuantLib::DerivedQuote
  - update, 398
- QuantLib::DirichletBC, 399
- QuantLib::DirichletBC
  - setTime, 399
- QuantLib::Discount, 400
- QuantLib::DiscrepancyStatistics, 401
- QuantLib::DiscreteAveragingAsianOption, 402
- QuantLib::DiscreteAveragingAsianOption
  - setupArguments, 403
- QuantLib::DiscreteAveragingAsianOption::arguments, 404
- QuantLib::DiscreteAveragingAsianOption::engine, 405
- QuantLib::DiscreteGeometricASO, 406
- QuantLib::DiscretizedAsset, 407
- QuantLib::DiscretizedAsset
  - adjustValues, 408
  - isOnTime, 408
  - mandatoryTimes, 408
  - postAdjustValues, 408
  - postAdjustValuesImpl, 409
  - preAdjustValues, 408
  - preAdjustValuesImpl, 409
  - reset, 408
- QuantLib::DiscretizedDiscountBond, 410
- QuantLib::DiscretizedDiscountBond
  - mandatoryTimes, 410
  - reset, 410
- QuantLib::DiscretizedOption, 411
- QuantLib::DiscretizedOption
  - mandatoryTimes, 411
  - postAdjustValuesImpl, 412
  - reset, 411
- QuantLib::Disposable, 413
- QuantLib::Dividend, 414
- QuantLib::Dividend
  - amount, 415
- QuantLib::DividendVanillaOption, 416
- QuantLib::DividendVanillaOption
  - setupArguments, 417
- QuantLib::DividendVanillaOption::arguments, 418
- QuantLib::DividendVanillaOption::engine, 419
- QuantLib::DKKCurrency, 420
- QuantLib::DKKLibor, 421
- QuantLib::DMinus, 422
- QuantLib::DownRounding, 423
- QuantLib::DPlus, 424
- QuantLib::DPlusDMinus, 425
- QuantLib::DriftCalculator, 426
- QuantLib::DriftCalculator
  - computePlain, 426
  - computeReduced, 426
  - DriftCalculator, 426
- QuantLib::DriftTermStructure, 427
- QuantLib::Duration, 429
- QuantLib::DZero, 430
- QuantLib::EarlyExercise, 431
- QuantLib::EarlyExercisePathPricer, 432
- QuantLib::EEKCurrency, 433
- QuantLib::EndCriteria, 434
- QuantLib::EndCriteria
  - operator(), 435
- QuantLib::EqualJumpsBinomialTree, 436
- QuantLib::EqualProbabilitiesBinomialTree, 437
- QuantLib::Error, 438
- QuantLib::Error
  - ~Error, 438
  - Error, 438
- QuantLib::ErrorFunction, 439
- QuantLib::ESPCurrency, 440
- QuantLib::EulerDiscretization, 441
- QuantLib::EulerDiscretization
  - covariance, 442
  - diffusion, 441
  - drift, 441
  - variance, 442
- QuantLib::EURCurrency, 443
- QuantLib::Euribor, 444
- QuantLib::Euribor10M, 445
- QuantLib::Euribor11M, 446
- QuantLib::Euribor1M, 447
- QuantLib::Euribor1Y, 448
- QuantLib::Euribor2M, 449
- QuantLib::Euribor2W, 450
- QuantLib::Euribor365, 451
- QuantLib::Euribor365\_10M, 452
- QuantLib::Euribor365\_11M, 453
- QuantLib::Euribor365\_1M, 454
- QuantLib::Euribor365\_1Y, 455
- QuantLib::Euribor365\_2M, 456
- QuantLib::Euribor365\_2W, 457
- QuantLib::Euribor365\_3M, 458
- QuantLib::Euribor365\_3W, 459
- QuantLib::Euribor365\_4M, 460
- QuantLib::Euribor365\_5M, 461
- QuantLib::Euribor365\_6M, 462
- QuantLib::Euribor365\_7M, 463
- QuantLib::Euribor365\_8M, 464
- QuantLib::Euribor365\_9M, 465
- QuantLib::Euribor365\_SW, 466
- QuantLib::Euribor3M, 467

- QuantLib::Euribor3W, [468](#)
- QuantLib::Euribor4M, [469](#)
- QuantLib::Euribor5M, [470](#)
- QuantLib::Euribor6M, [471](#)
- QuantLib::Euribor7M, [472](#)
- QuantLib::Euribor8M, [473](#)
- QuantLib::Euribor9M, [474](#)
- QuantLib::EuriborSW, [475](#)
- QuantLib::EuriborSwapFixA, [476](#)
- QuantLib::EuriborSwapFixA10Y, [477](#)
- QuantLib::EuriborSwapFixA12Y, [478](#)
- QuantLib::EuriborSwapFixA15Y, [479](#)
- QuantLib::EuriborSwapFixA1Y, [480](#)
- QuantLib::EuriborSwapFixA20Y, [481](#)
- QuantLib::EuriborSwapFixA25Y, [482](#)
- QuantLib::EuriborSwapFixA2Y, [483](#)
- QuantLib::EuriborSwapFixA30Y, [484](#)
- QuantLib::EuriborSwapFixA3Y, [485](#)
- QuantLib::EuriborSwapFixA4Y, [486](#)
- QuantLib::EuriborSwapFixA5Y, [487](#)
- QuantLib::EuriborSwapFixA6Y, [488](#)
- QuantLib::EuriborSwapFixA7Y, [489](#)
- QuantLib::EuriborSwapFixA8Y, [490](#)
- QuantLib::EuriborSwapFixA9Y, [491](#)
- QuantLib::EuriborSwapFixIFR, [492](#)
- QuantLib::EuriborSwapFixIFR10Y, [493](#)
- QuantLib::EuriborSwapFixIFR12Y, [494](#)
- QuantLib::EuriborSwapFixIFR15Y, [495](#)
- QuantLib::EuriborSwapFixIFR1Y, [496](#)
- QuantLib::EuriborSwapFixIFR20Y, [497](#)
- QuantLib::EuriborSwapFixIFR25Y, [498](#)
- QuantLib::EuriborSwapFixIFR2Y, [499](#)
- QuantLib::EuriborSwapFixIFR30Y, [500](#)
- QuantLib::EuriborSwapFixIFR3Y, [501](#)
- QuantLib::EuriborSwapFixIFR4Y, [502](#)
- QuantLib::EuriborSwapFixIFR5Y, [503](#)
- QuantLib::EuriborSwapFixIFR6Y, [504](#)
- QuantLib::EuriborSwapFixIFR7Y, [505](#)
- QuantLib::EuriborSwapFixIFR8Y, [506](#)
- QuantLib::EuriborSwapFixIFR9Y, [507](#)
- QuantLib::EURLibor, [508](#)
- QuantLib::EURLibor10M, [509](#)
- QuantLib::EURLibor11M, [510](#)
- QuantLib::EURLibor1M, [511](#)
- QuantLib::EURLibor1Y, [512](#)
- QuantLib::EURLibor2M, [513](#)
- QuantLib::EURLibor2W, [514](#)
- QuantLib::EURLibor3M, [515](#)
- QuantLib::EURLibor4M, [516](#)
- QuantLib::EURLibor5M, [517](#)
- QuantLib::EURLibor6M, [518](#)
- QuantLib::EURLibor7M, [519](#)
- QuantLib::EURLibor8M, [520](#)
- QuantLib::EURLibor9M, [521](#)
- QuantLib::EURLiborSW, [522](#)
- QuantLib::EurliborSwapFixA, [523](#)
- QuantLib::EurliborSwapFixA10Y, [524](#)
- QuantLib::EurliborSwapFixA12Y, [525](#)
- QuantLib::EurliborSwapFixA15Y, [526](#)
- QuantLib::EurliborSwapFixA1Y, [527](#)
- QuantLib::EurliborSwapFixA20Y, [528](#)
- QuantLib::EurliborSwapFixA25Y, [529](#)
- QuantLib::EurliborSwapFixA2Y, [530](#)
- QuantLib::EurliborSwapFixA30Y, [531](#)
- QuantLib::EurliborSwapFixA3Y, [532](#)
- QuantLib::EurliborSwapFixA4Y, [533](#)
- QuantLib::EurliborSwapFixA5Y, [534](#)
- QuantLib::EurliborSwapFixA6Y, [535](#)
- QuantLib::EurliborSwapFixA7Y, [536](#)
- QuantLib::EurliborSwapFixA8Y, [537](#)
- QuantLib::EurliborSwapFixA9Y, [538](#)
- QuantLib::EurliborSwapFixB, [539](#)
- QuantLib::EurliborSwapFixB10Y, [540](#)
- QuantLib::EurliborSwapFixB12Y, [541](#)
- QuantLib::EurliborSwapFixB15Y, [542](#)
- QuantLib::EurliborSwapFixB1Y, [543](#)
- QuantLib::EurliborSwapFixB20Y, [544](#)
- QuantLib::EurliborSwapFixB25Y, [545](#)
- QuantLib::EurliborSwapFixB2Y, [546](#)
- QuantLib::EurliborSwapFixB30Y, [547](#)
- QuantLib::EurliborSwapFixB3Y, [548](#)
- QuantLib::EurliborSwapFixB4Y, [549](#)
- QuantLib::EurliborSwapFixB5Y, [550](#)
- QuantLib::EurliborSwapFixB6Y, [551](#)
- QuantLib::EurliborSwapFixB7Y, [552](#)
- QuantLib::EurliborSwapFixB8Y, [553](#)
- QuantLib::EurliborSwapFixB9Y, [554](#)
- QuantLib::EurliborSwapFixIFR, [555](#)
- QuantLib::EurliborSwapFixIFR10Y, [556](#)
- QuantLib::EurliborSwapFixIFR12Y, [557](#)
- QuantLib::EurliborSwapFixIFR15Y, [558](#)
- QuantLib::EurliborSwapFixIFR1Y, [559](#)
- QuantLib::EurliborSwapFixIFR20Y, [560](#)
- QuantLib::EurliborSwapFixIFR25Y, [561](#)
- QuantLib::EurliborSwapFixIFR2Y, [562](#)
- QuantLib::EurliborSwapFixIFR30Y, [563](#)
- QuantLib::EurliborSwapFixIFR3Y, [564](#)
- QuantLib::EurliborSwapFixIFR4Y, [565](#)
- QuantLib::EurliborSwapFixIFR5Y, [566](#)
- QuantLib::EurliborSwapFixIFR6Y, [567](#)
- QuantLib::EurliborSwapFixIFR7Y, [568](#)
- QuantLib::EurliborSwapFixIFR8Y, [569](#)
- QuantLib::EurliborSwapFixIFR9Y, [570](#)
- QuantLib::EuropeanExercise, [571](#)
- QuantLib::EuropeanOption, [572](#)
- QuantLib::Event, [573](#)
- QuantLib::Event
  - hasOccurred, [573](#)

- QuantLib::EvolutionDescription, 575
- QuantLib::ExchangeRate, 576
  - Derived, 576
  - Direct, 576
- QuantLib::ExchangeRate
  - ExchangeRate, 577
  - Type, 576
- QuantLib::ExchangeRateManager, 578
- QuantLib::ExchangeRateManager
  - add, 578
  - lookup, 578
- QuantLib::Exercise, 580
- QuantLib::ExplicitEuler, 581
- QuantLib::ExtendedCoxIngersollRoss, 583
- QuantLib::ExtendedCoxIngersollRoss::Dynamics, 585
- QuantLib::ExtendedCoxIngersollRoss::FittingParameters, 586
- QuantLib::ExtendedDiscountCurve, 587
- QuantLib::ExtendedDiscountCurve
  - compoundForwardImpl, 588
  - update, 588
  - zeroYieldImpl, 588
- QuantLib::Extrapolator, 589
- QuantLib::Factorial, 590
- QuantLib::FalsePosition, 591
- QuantLib::FaureRsg, 592
- QuantLib::FDAmericanCondition, 593
- QuantLib::FDBermudanEngine, 594
- QuantLib::FDDividendEngineMerton73, 595
- QuantLib::FDDividendEngineShiftScale, 596
- QuantLib::FDEuropeanEngine, 597
- QuantLib::FDStepConditionEngine, 598
- QuantLib::FIMCurrency, 599
- QuantLib::FiniteDifferenceModel, 600
- QuantLib::FiniteDifferenceModel
  - rollback, 600
- QuantLib::Finland, 601
- QuantLib::FixedCouponBond, 602
- QuantLib::FixedCouponBond
  - FixedCouponBond, 603
- QuantLib::FixedCouponBondForward, 604
- QuantLib::FixedCouponBondForward
  - FixedCouponBondForward, 605
  - performCalculations, 606
  - spotIncome, 606
- QuantLib::FixedCouponBondHelper, 607
- QuantLib::FixedCouponBondHelper
  - latestDate, 608
  - setTermStructure, 608
- QuantLib::FixedDividend, 609
- QuantLib::FixedDividend
  - amount, 609
- QuantLib::FixedRateCoupon, 611
- QuantLib::FixedRateCoupon
  - amount, 612
- QuantLib::FlatForward, 613
- QuantLib::FlatForward
  - update, 614
- QuantLib::FloatingRateBond, 615
- QuantLib::FloatingRateBond
  - FloatingRateBond, 616
- QuantLib::FloatingRateCoupon, 617
- QuantLib::FloatingRateCoupon
  - amount, 619
  - update, 619
- QuantLib::FloatingTypePayoff, 620
- QuantLib::Floor, 621
- QuantLib::FloorTruncation, 622
- QuantLib::Forward, 623
- QuantLib::Forward
  - forwardValue, 625
  - impliedYield, 625
  - incomeDiscountCurve\_, 625
  - performCalculations, 625
  - underlyingIncome\_, 625
  - underlyingSpotValue\_, 625
  - valueDate\_, 625
- QuantLib::ForwardEngine, 626
- QuantLib::ForwardFlat, 627
- QuantLib::ForwardFlatInterpolation, 628
- QuantLib::ForwardFlatInterpolation
  - ForwardFlatInterpolation, 628
- QuantLib::ForwardMeasureProcess, 629
- QuantLib::ForwardMeasureProcess1D, 630
- QuantLib::ForwardOptionArguments, 631
- QuantLib::ForwardPerformanceEngine, 632
- QuantLib::ForwardRate, 633
- QuantLib::ForwardRateAgreement, 634
- QuantLib::ForwardRateAgreement
  - isExpired, 635
  - performCalculations, 636
  - settlementDate, 635
  - spotIncome, 636
  - spotValue, 636
- QuantLib::ForwardRateIpcEvolver, 637
- QuantLib::ForwardRatePcEvolver, 638
- QuantLib::ForwardRateStructure, 639
- QuantLib::ForwardRateStructure
  - discountImpl, 639
  - zeroYieldImpl, 639
- QuantLib::ForwardSpreadedTermStructure, 641
- QuantLib::ForwardTypePayoff, 643
- QuantLib::ForwardVanillaOption, 644
- QuantLib::ForwardVanillaOption
  - fetchResults, 645
  - setupArguments, 645

- QuantLib::FractionalDividend, 646
- QuantLib::FractionalDividend
  - amount, 647
- QuantLib::FraRateHelper, 648
- QuantLib::FRFCurrency, 649
- QuantLib::FuturesRateHelper, 650
- QuantLib::G2, 651
- QuantLib::G2::FittingParameter, 653
- QuantLib::G2ForwardProcess, 654
- QuantLib::G2ForwardProcess
  - covariance, 655
  - expectation, 655
  - stdDeviation, 655
- QuantLib::G2Process, 656
- QuantLib::G2Process
  - covariance, 657
  - expectation, 657
  - stdDeviation, 657
- QuantLib::G2SwaptionEngine, 658
- QuantLib::GammaFunction, 659
- QuantLib::GapPayoff, 660
- QuantLib::Garch11, 661
- QuantLib::GarmanKlassAbstract, 662
- QuantLib::GarmanKlassOpenClose, 663
- QuantLib::GarmanKohlagenProcess, 664
- QuantLib::GaussChebyshev2thIntegration, 665
- QuantLib::GaussChebyshevIntegration, 666
- QuantLib::GaussGegenbauerIntegration, 667
- QuantLib::GaussHermiteIntegration, 668
- QuantLib::GaussHermitePolynomial, 669
- QuantLib::GaussHyperbolicIntegration, 670
- QuantLib::GaussHyperbolicPolynomial, 671
- QuantLib::GaussianOrthogonalPolynomial, 672
- QuantLib::GaussianQuadrature, 673
- QuantLib::GaussJacobiIntegration, 674
- QuantLib::GaussJacobiPolynomial, 675
- QuantLib::GaussLaguerreIntegration, 676
- QuantLib::GaussLaguerrePolynomial, 677
- QuantLib::GaussLegendreIntegration, 678
- QuantLib::GBPCurrency, 679
- QuantLib::GBPLibor, 680
- QuantLib::GeneralizedBlackScholesProcess, 681
- QuantLib::GeneralizedBlackScholesProcess
  - apply, 682
  - diffusion, 682
  - drift, 682
  - time, 682
  - update, 682
- QuantLib::GeneralStatistics, 683
- QuantLib::GeneralStatistics
  - add, 685
  - errorEstimate, 684
  - expectationValue, 685
  - kurtosis, 684
  - max, 685
  - mean, 684
  - min, 684
  - percentile, 685
  - skewness, 684
  - standardDeviation, 684
  - topPercentile, 685
  - variance, 684
- QuantLib::GenericEngine, 686
- QuantLib::GenericGaussianStatistics, 688
- QuantLib::GenericGaussianStatistics
  - gaussianDownsideDeviation, 689
  - gaussianDownsideVariance, 688
  - gaussianExpectedShortfall, 689
  - gaussianPercentile, 689
  - gaussianPotentialUpside, 689
  - gaussianRegret, 689
  - gaussianTopPercentile, 689
  - gaussianValueAtRisk, 689
- QuantLib::GenericModelEngine, 691
- QuantLib::GenericModelEngine
  - update, 691
- QuantLib::GenericRiskStatistics, 692
- QuantLib::GenericRiskStatistics
  - averageShortfall, 694
  - downsideDeviation, 693
  - downsideVariance, 693
  - expectedShortfall, 693
  - potentialUpside, 693
  - regret, 693
  - semiDeviation, 693
  - semiVariance, 693
  - shortfall, 694
  - valueAtRisk, 693
- QuantLib::GenericSequenceStatistics, 695
- QuantLib::GeometricBrownianMotionProcess, 697
- QuantLib::Germany, 698
  - Eurex, 700
  - FrankfurtStockExchange, 700
  - Settlement, 700
  - Xetra, 700
- QuantLib::Germany
  - Market, 700
- QuantLib::GRDCurrency, 701
- QuantLib::Greeks, 702
- QuantLib::HaltonRsg, 703
- QuantLib::Handle, 704
- QuantLib::Handle
  - Handle, 705
  - linkTo, 705
- QuantLib::HestonModel, 706

- QuantLib::HestonModelHelper, 707
- QuantLib::HestonProcess, 708
- QuantLib::HestonProcess
  - apply, 709
  - time, 709
- QuantLib::HKDCurrency, 710
- QuantLib::HongKong, 711
  - HKEEx, 712
- QuantLib::HongKong
  - Market, 712
- QuantLib::HUFCurrency, 713
- QuantLib::HullWhite, 714
- QuantLib::HullWhite
  - convexityBias, 715
- QuantLib::HullWhite::Dynamics, 716
- QuantLib::HullWhite::FittingParameter, 717
- QuantLib::HullWhiteForwardProcess, 718
- QuantLib::HullWhiteForwardProcess
  - expectation, 719
  - stdDeviation, 719
  - variance, 719
- QuantLib::HullWhiteProcess, 720
- QuantLib::HullWhiteProcess
  - expectation, 721
  - stdDeviation, 721
  - variance, 721
- QuantLib::Hungary, 722
- QuantLib::Iceland, 723
  - ICEX, 724
- QuantLib::Iceland
  - Market, 724
- QuantLib::IEPCurrency, 725
- QuantLib::ILSCurrency, 726
- QuantLib::IMM, 727
- QuantLib::ImplicitEuler, 728
- QuantLib::ImpliedTermStructure, 729
- QuantLib::ImpliedVolTermStructure, 731
- QuantLib::InArrearIndexedCoupon, 733
- QuantLib::IncrementalStatistics, 735
- QuantLib::IncrementalStatistics
  - add, 737
  - addSequence, 737
  - downsideDeviation, 737
  - downsideVariance, 737
  - errorEstimate, 736
  - kurtosis, 736
  - max, 737
  - mean, 736
  - min, 737
  - skewness, 736
  - standardDeviation, 736
  - variance, 736
- QuantLib::Index, 738
- QuantLib::Index
  - addFixing, 739
  - addFixings, 739
  - fixing, 739
  - name, 738
- QuantLib::IndexManager, 740
- QuantLib::India, 741
  - NSE, 742
- QuantLib::India
  - Market, 742
- QuantLib::Indonesia, 743
  - BEJ, 744
- QuantLib::Indonesia
  - Market, 744
- QuantLib::INRCurrency, 745
- QuantLib::Instrument, 746
- QuantLib::Instrument
  - calculate, 747
  - fetchResults, 747
  - performCalculations, 748
  - setPricingEngine, 747
  - setupArguments, 747
  - setupExpired, 748
- QuantLib::IntegralEngine, 749
- QuantLib::InterestRate, 750
- QuantLib::InterestRate
  - compoundFactor, 751
  - discountFactor, 751
  - equivalentRate, 752
  - impliedRate, 751, 752
- QuantLib::InterestRateIndex, 753
- QuantLib::InterestRateIndex
  - fixing, 754
  - name, 754
  - update, 754
- QuantLib::InterpolatedDiscountCurve, 755
- QuantLib::InterpolatedForwardCurve, 757
- QuantLib::InterpolatedForwardCurve
  - zeroYieldImpl, 758
- QuantLib::InterpolatedZeroCurve, 759
- QuantLib::Interpolation, 761
- QuantLib::Interpolation2D, 762
- QuantLib::Interpolation2D::templateImpl, 764
- QuantLib::Interpolation2DImpl, 765
- QuantLib::Interpolation::templateImpl, 766
- QuantLib::InterpolationImpl, 767
- QuantLib::IntervalPrice, 768
- QuantLib::InverseCumulativeNormal, 769
- QuantLib::InverseCumulativePoisson, 770
- QuantLib::InverseCumulativeRng, 771
- QuantLib::InverseCumulativeRsg, 772
- QuantLib::IQDCurrency, 773
- QuantLib::IRRCurrency, 774
- QuantLib::ISKCurrency, 775
- QuantLib::Italy, 776



- Exchange, [777](#)
- Settlement, [777](#)
- QuantLib::Italy
  - Market, [777](#)
- QuantLib::ITLCurrency, [778](#)
- QuantLib::JamshidianSwaptionEngine, [779](#)
- QuantLib::Japan, [780](#)
- QuantLib::JarrowRudd, [782](#)
- QuantLib::Jibar, [783](#)
- QuantLib::JointCalendar, [784](#)
- QuantLib::JPYCurrency, [785](#)
- QuantLib::JPYLibor, [786](#)
- QuantLib::JumpDiffusionEngine, [787](#)
- QuantLib::JuQuadraticApproximationEngine, [788](#)
- QuantLib::KnuthUniformRng, [789](#)
- QuantLib::KnuthUniformRng
  - KnuthUniformRng, [789](#)
  - next, [789](#)
- QuantLib::KronrodIntegral, [790](#)
- QuantLib::KRWCurrency, [791](#)
- QuantLib::KWDCurrency, [792](#)
- QuantLib::Lattice, [793](#)
- QuantLib::Lattice
  - partialRollback, [794](#)
  - rollback, [794](#)
- QuantLib::Lattice1D, [795](#)
- QuantLib::Lattice2D, [796](#)
- QuantLib::LatticeShortRateModelEngine, [797](#)
- QuantLib::LatticeShortRateModelEngine
  - update, [797](#)
- QuantLib::LazyObject, [798](#)
- QuantLib::LazyObject
  - calculate, [799](#)
  - freeze, [799](#)
  - performCalculations, [799](#)
  - recalculate, [799](#)
  - unfreeze, [799](#)
  - update, [798](#)
- QuantLib::LeastSquareFunction, [800](#)
- QuantLib::LeastSquareProblem, [801](#)
- QuantLib::LeastSquareProblem
  - targetValueAndGradient, [801](#)
- QuantLib::LecuyerUniformRng, [802](#)
- QuantLib::LecuyerUniformRng
  - LecuyerUniformRng, [802](#)
  - next, [802](#)
- QuantLib::LeisenReimer, [803](#)
- QuantLib::LevenbergMarquardt, [804](#)
- QuantLib::LevenbergMarquardt
  - LevenbergMarquardt, [804](#)
- QuantLib::LexicographicalView, [805](#)
- QuantLib::LfmCovarianceParameterization, [807](#)
- QuantLib::LfmCovarianceProxy, [808](#)
- QuantLib::LfmHullWhiteParameterization, [809](#)
- QuantLib::LfmSwaptionEngine, [810](#)
- QuantLib::Libor, [811](#)
- QuantLib::LiborForwardModel, [812](#)
- QuantLib::LiborForwardModelProcess, [814](#)
- QuantLib::LiborForwardModelProcess
  - apply, [815](#)
  - covariance, [815](#)
  - evolve, [815](#)
- QuantLib::Linear, [816](#)
- QuantLib::LinearInterpolation, [817](#)
- QuantLib::LinearInterpolation
  - LinearInterpolation, [817](#)
- QuantLib::LinearLeastSquaresRegression, [818](#)
- QuantLib::LineSearch, [819](#)
- QuantLib::Link, [821](#)
- QuantLib::Link
  - Link, [822](#)
  - linkTo, [822](#)
- QuantLib::LmConstWrapperVolatilityModel, [823](#)
- QuantLib::LmCorrelationModel, [824](#)
- QuantLib::LmExponentialCorrelationModel, [825](#)
- QuantLib::LmExtLinearExponentialVolModel, [826](#)
- QuantLib::LmLinearExponentialCorrelationModel, [827](#)
- QuantLib::LmLinearExponentialVolatilityModel, [828](#)
- QuantLib::LmVolatilityModel, [829](#)
- QuantLib::LocalConstantVol, [830](#)
- QuantLib::LocalVolatilityEstimator, [832](#)
- QuantLib::LocalVolCurve, [833](#)
- QuantLib::LocalVolCurve
  - localVolImpl, [834](#)
- QuantLib::LocalVolSurface, [835](#)
- QuantLib::LocalVolTermStructure, [837](#)
- QuantLib::LocalVolTermStructure
  - LocalVolTermStructure, [838](#)
- QuantLib::LogLinear, [839](#)
- QuantLib::LogLinearInterpolation, [840](#)
- QuantLib::LogLinearInterpolation
  - LogLinearInterpolation, [840](#)
- QuantLib::LongstaffSchwartzPathPricer, [841](#)
- QuantLib::LTLCurrency, [842](#)
- QuantLib::LUFCurrency, [843](#)
- QuantLib::LVLCurrency, [844](#)
- QuantLib::MakeMCAmericanEngine, [845](#)
- QuantLib::MakeMCDigitalEngine, [846](#)
- QuantLib::MakeMCEuropeanEngine, [847](#)

- QuantLib::MakeMCEuropeanHestonEngine, 848
- QuantLib::MakeMCHullWhiteCapFloorEngine, 849
- QuantLib::MakeMCVarianceSwapEngine, 850
- QuantLib::MakeSchedule, 851
- QuantLib::MakeVanillaSwap, 852
- QuantLib::MarketModelComposite, 853
- QuantLib::MarketModelEvolver, 855
- QuantLib::MarketModelMultiProduct, 856
- QuantLib::Matrix, 857
- QuantLib::Matrix
  - operator+=, 859
  - pseudoSqrt, 859
  - rankReducedSqrt, 860
- QuantLib::MCAmericanBasketEngine, 861
- QuantLib::MCAmericanEngine, 862
- QuantLib::MCBarrierEngine, 863
- QuantLib::MCBasketEngine, 865
- QuantLib::McCliquetOption, 867
- QuantLib::MCDigitalEngine, 868
- QuantLib::MCDiscreteArithmeticAPEngine, 869
- QuantLib::MCDiscreteArithmeticASO, 870
- QuantLib::MCDiscreteAveragingAsianEngine, 871
- QuantLib::MCDiscreteGeometricAPEngine, 873
- QuantLib::MCEuropeanEngine, 874
- QuantLib::MCEuropeanHestonEngine, 875
- QuantLib::McEverest, 876
- QuantLib::McHimalaya, 877
- QuantLib::MCHullWhiteCapFloorEngine, 878
- QuantLib::MCHullWhiteCapFloorEngine
  - update, 878
- QuantLib::MCLongstaffSchwartzEngine, 879
- QuantLib::McMaxBasket, 881
- QuantLib::McPagoda, 882
- QuantLib::McPerformanceOption, 883
- QuantLib::McPricer, 884
- QuantLib::McSimulation, 885
- QuantLib::MCVanillaEngine, 887
- QuantLib::MCVarianceSwapEngine, 888
- QuantLib::MersenneTwisterUniformRng, 890
- QuantLib::MersenneTwisterUniformRng
  - MersenneTwisterUniformRng, 890
  - next, 890
- QuantLib::Merton76Process, 891
- QuantLib::Merton76Process
  - apply, 892
  - time, 892
- QuantLib::Mexico, 893
  - BMV, 894
- QuantLib::Mexico
  - Market, 894
- QuantLib::MixedScheme, 895
- QuantLib::Money, 897
  - AutomatedConversion, 898
  - BaseCurrencyConversion, 898
  - NoConversion, 898
- QuantLib::Money
  - ConversionType, 898
- QuantLib::MonotonicCubicSpline, 899
- QuantLib::MonotonicCubicSpline
  - MonotonicCubicSpline, 899
- QuantLib::MonteCarloModel, 900
- QuantLib::MoreGreeks, 901
- QuantLib::MoroInverseCumulativeNormal, 902
- QuantLib::MTBrownianGenerator, 903
- QuantLib::MTLCurrency, 904
- QuantLib::MultiAssetOption, 905
- QuantLib::MultiAssetOption
  - fetchResults, 906
  - setupArguments, 906
  - setupExpired, 906
- QuantLib::MultiAssetOption::arguments, 907
- QuantLib::MultiAssetOption::results, 908
- QuantLib::MultiCubicSpline, 909
- QuantLib::MultiPath, 910
- QuantLib::MultiPathGenerator, 911
- QuantLib::MultiProductComposite, 912
- QuantLib::MultiProductMultiStep, 913
- QuantLib::MultiProductOneStep, 914
- QuantLib::MultiVariate, 915
- QuantLib::MXNCurrency, 916
- QuantLib::NaturalCubicSpline, 917
- QuantLib::NaturalCubicSpline
  - NaturalCubicSpline, 917
- QuantLib::NaturalMonotonicCubicSpline, 918
- QuantLib::NaturalMonotonicCubicSpline
  - NaturalMonotonicCubicSpline, 918
- QuantLib::NeumannBC, 919
- QuantLib::NeumannBC
  - setTime, 919
- QuantLib::Newton, 920
- QuantLib::NewtonSafe, 921
- QuantLib::NewZealand, 922
- QuantLib::NLGCurrency, 923
- QuantLib::NoConstraint, 924
- QuantLib::NOKCurrency, 925
- QuantLib::NonLinearLeastSquare, 926
- QuantLib::NormalDistribution, 927
- QuantLib::Norway, 928
- QuantLib::NPRCurrency, 929
- QuantLib::Null, 930
- QuantLib::NullCalendar, 931
- QuantLib::NullCondition, 932

- QuantLib::NullParameter, 933
- QuantLib::NumericalMethod, 934
- QuantLib::NumericalMethod
  - partialRollback, 934
  - rollback, 934
- QuantLib::NZDCurrency, 936
- QuantLib::NZDLibor, 937
- QuantLib::Observable, 938
- QuantLib::Observable
  - notifyObservers, 939
  - operator=, 939
- QuantLib::ObservableValue, 940
- QuantLib::Observer, 941
- QuantLib::Observer
  - update, 942
- QuantLib::OneAssetOption, 943
- QuantLib::OneAssetOption
  - fetchResults, 945
  - impliedVolatility, 944
  - setupArguments, 944
  - setupExpired, 945
- QuantLib::OneAssetOption::arguments, 946
- QuantLib::OneAssetOption::results, 947
- QuantLib::OneAssetStrikedOption, 948
- QuantLib::OneAssetStrikedOption
  - fetchResults, 949
  - setupArguments, 948
  - setupExpired, 949
- QuantLib::OneDayCounter, 950
- QuantLib::OneFactorAffineModel, 951
- QuantLib::OneFactorModel, 952
- QuantLib::OneFactorModel::ShortRateDynamics, 953
- QuantLib::OneFactorModel::ShortRateTree, 954
- QuantLib::OperatorFactory, 955
- QuantLib::OptimizationMethod, 956
- QuantLib::Option, 958
- QuantLib::Option::arguments, 959
- QuantLib::OrnsteinUhlenbeckProcess, 960
- QuantLib::OrnsteinUhlenbeckProcess
  - expectation, 960
  - stdDeviation, 960
  - variance, 961
- QuantLib::Parameter, 962
- QuantLib::ParameterImpl, 963
- QuantLib::ParCoupon, 964
- QuantLib::Path, 965
- QuantLib::PathGenerator, 966
- QuantLib::PathPricer, 967
- QuantLib::Payoff, 968
- QuantLib::PercentageStrikePayoff, 969
- QuantLib::Period, 970
- QuantLib::PiecewiseConstantParameter, 971
- QuantLib::PiecewiseYieldCurve, 972
- QuantLib::PiecewiseYieldCurve
  - update, 973
- QuantLib::PiecewiseZeroSpreadedTermStructure, 974
- QuantLib::PiecewiseZeroSpreadedTermStructure
  - update, 975
- QuantLib::PKRCurrency, 976
- QuantLib::PlainVanillaPayoff, 977
- QuantLib::PLNCurrency, 978
- QuantLib::PoissonDistribution, 979
- QuantLib::Poland, 980
- QuantLib::PositiveConstraint, 981
- QuantLib::PriceCurve, 982
- QuantLib::PricingEngine, 983
- QuantLib::PrimeNumbers, 984
- QuantLib::Problem, 985
- QuantLib::PTECurrency, 987
- QuantLib::QuantoEngine, 988
- QuantLib::QuantoEngine
  - underlyingArgs, 989
- QuantLib::QuantoForwardVanillaOption, 990
- QuantLib::QuantoForwardVanillaOption
  - setupArguments, 991
- QuantLib::QuantoOptionArguments, 992
- QuantLib::QuantoOptionResults, 993
- QuantLib::QuantoTermStructure, 994
- QuantLib::QuantoVanillaOption, 996
- QuantLib::QuantoVanillaOption
  - fetchResults, 997
  - setupArguments, 997
  - setupExpired, 997
- QuantLib::Quote, 998
- QuantLib::RandomizedLDS, 999
- QuantLib::RandomizedLDS
  - nextRandomizer, 1000
- QuantLib::RandomSequenceGenerator, 1001
- QuantLib::RateHelper, 1002
- QuantLib::RateHelper
  - earliestDate, 1003
  - latestDate, 1003
  - setTermStructure, 1003
  - update, 1003
- QuantLib::RelativeDateRateHelper, 1004
- QuantLib::RelativeDateRateHelper
  - update, 1004
- QuantLib::ReplicatingVarianceSwapEngine, 1005
- QuantLib::Results, 1006
- QuantLib::Ridder, 1007
- QuantLib::ROLCurrency, 1008
- QuantLib::RONCurrency, 1009
- QuantLib::Rounding, 1010



- Ceiling, [1011](#)
- Closest, [1011](#)
- Down, [1011](#)
- Floor, [1011](#)
- None, [1011](#)
- Up, [1011](#)
- QuantLib::Rounding
  - Rounding, [1011](#)
  - Type, [1010](#)
- QuantLib::SABRInterpolation, [1012](#)
- QuantLib::SalvagingAlgorithm, [1013](#)
- QuantLib::Sample, [1014](#)
- QuantLib::SampledCurve, [1015](#)
- QuantLib::SampledCurve
  - firstDerivativeAtCenter, [1016](#)
  - secondDerivativeAtCenter, [1016](#)
  - valueAtCenter, [1016](#)
- QuantLib::SARCurrency, [1017](#)
- QuantLib::SaudiArabia, [1018](#)
- QuantLib::Schedule, [1019](#)
- QuantLib::Schedule
  - Schedule, [1020](#)
- QuantLib::Secant, [1021](#)
- QuantLib::SeedGenerator, [1022](#)
- QuantLib::SegmentIntegral, [1023](#)
- QuantLib::SEKCurrency, [1024](#)
- QuantLib::Settings, [1025](#)
- QuantLib::Settings
  - evaluationDate, [1025](#)
- QuantLib::Settlement, [1027](#)
- QuantLib::SGDCurrency, [1028](#)
- QuantLib::Short, [1029](#)
- QuantLib::Short
  - amount, [1029](#)
- QuantLib::Short< ParCoupon >, [1030](#)
- QuantLib::ShortRateModel, [1031](#)
- QuantLib::ShoutCondition, [1032](#)
- QuantLib::SimpleCashFlow, [1033](#)
- QuantLib::SimpleCashFlow
  - amount, [1033](#)
- QuantLib::SimpleDayCounter, [1034](#)
- QuantLib::SimpleLocalEstimator, [1035](#)
- QuantLib::SimpleQuote, [1036](#)
- QuantLib::Simplex, [1037](#)
- QuantLib::Simplex
  - Simplex, [1037](#)
- QuantLib::SimpsonIntegral, [1038](#)
- QuantLib::Singapore, [1039](#)
- QuantLib::Singapore
  - SGX, [1040](#)
- QuantLib::Singapore
  - Market, [1040](#)
- QuantLib::SingleAssetOption, [1041](#)
- QuantLib::SingleAssetOption
  - impliedVolatility, [1042](#)
- QuantLib::SingleProductComposite, [1043](#)
- QuantLib::Singleton, [1044](#)
- QuantLib::SingleVariate, [1045](#)
- QuantLib::SITCurrency, [1046](#)
- QuantLib::SKKCurrency, [1047](#)
- QuantLib::Slovakia, [1048](#)
- QuantLib::Slovakia
  - BSSE, [1049](#)
- QuantLib::Slovakia
  - Market, [1049](#)
- QuantLib::SmileSection, [1050](#)
- QuantLib::SobolRsg, [1051](#)
- QuantLib::SobolRsg
  - skipTo, [1052](#)
  - SobolRsg, [1052](#)
- QuantLib::SoftCallability, [1053](#)
- QuantLib::Solver1D, [1054](#)
- QuantLib::Solver1D
  - setMaxEvaluations, [1055](#)
  - solve, [1055](#)
- QuantLib::SouthAfrica, [1056](#)
- QuantLib::SouthKorea, [1057](#)
- QuantLib::SouthKorea
  - KRX, [1058](#)
- QuantLib::SouthKorea
  - Market, [1058](#)
- QuantLib::SquareRootProcess, [1059](#)
- QuantLib::StatsHolder, [1060](#)
- QuantLib::SteepestDescent, [1061](#)
- QuantLib::step\_iterator, [1062](#)
- QuantLib::StepCondition, [1063](#)
- QuantLib::StepConditionSet, [1064](#)
- QuantLib::StochasticProcess, [1065](#)
- QuantLib::StochasticProcess
  - apply, [1067](#)
  - covariance, [1066](#)
  - evolve, [1066](#)
  - expectation, [1066](#)
  - stdDeviation, [1066](#)
  - time, [1067](#)
  - update, [1067](#)
- QuantLib::StochasticProcess1D, [1068](#)
- QuantLib::StochasticProcess1D
  - apply, [1069](#)
  - evolve, [1069](#)
  - expectation, [1069](#)
  - stdDeviation, [1069](#)
  - variance, [1069](#)
- QuantLib::StochasticProcess1D::discretization, [1070](#)
- QuantLib::StochasticProcess::discretization, [1071](#)
- QuantLib::StochasticProcessArray, [1072](#)
- QuantLib::StochasticProcessArray
  - apply, [1073](#)
  - covariance, [1073](#)

- expectation, [1073](#)
  - stdDeviation, [1073](#)
  - time, [1073](#)
- QuantLib::Stock, [1074](#)
- QuantLib::Stock
  - performCalculations, [1074](#)
- QuantLib::StrikedTypePayoff, [1075](#)
- QuantLib::StulzEngine, [1076](#)
- QuantLib::SuperSharePayoff, [1077](#)
- QuantLib::SVD, [1078](#)
- QuantLib::Swap, [1079](#)
- QuantLib::Swap
  - performCalculations, [1080](#)
  - setupExpired, [1080](#)
  - Swap, [1080](#)
- QuantLib::SwapIndex, [1081](#)
- QuantLib::SwapIndex
  - underlyingSwap, [1082](#)
- QuantLib::SwapRateHelper, [1083](#)
- QuantLib::SwapRateHelper
  - setTermStructure, [1084](#)
- QuantLib::Swaption, [1085](#)
- QuantLib::Swaption
  - setupArguments, [1086](#)
- QuantLib::Swaption::arguments, [1087](#)
- QuantLib::Swaption::engine, [1088](#)
- QuantLib::Swaption::results, [1089](#)
- QuantLib::SwaptionConstantVolatility, [1090](#)
- QuantLib::SwaptionHelper, [1092](#)
- QuantLib::SwaptionVolatilityCube, [1093](#)
- QuantLib::SwaptionVolatilityCubeBySabr, [1095](#)
- QuantLib::SwaptionVolatilityMatrix, [1097](#)
- QuantLib::SwaptionVolatilityStructure, [1099](#)
- QuantLib::SwaptionVolatilityStructure
  - SwaptionVolatilityStructure, [1101](#)
- QuantLib::Sweden, [1102](#)
- QuantLib::Switzerland, [1103](#)
- QuantLib::SymmetricSchurDecomposition, [1104](#)
- QuantLib::SymmetricSchurDecomposition
  - SymmetricSchurDecomposition, [1104](#)
- QuantLib::TabulatedGaussLegendre, [1105](#)
- QuantLib::Taiwan, [1106](#)
  - TSEC, [1107](#)
- QuantLib::Taiwan
  - Market, [1107](#)
- QuantLib::TARGET, [1108](#)
- QuantLib::TermStructure, [1109](#)
- QuantLib::TermStructure
  - TermStructure, [1110](#)
  - update, [1110](#)
- QuantLib::TermStructureConsistentModel, [1111](#)
- QuantLib::TermStructureFittingParameter, [1112](#)
- QuantLib::THBCurrency, [1113](#)
- QuantLib::Thirty360, [1114](#)
- QuantLib::Tian, [1115](#)
- QuantLib::Tibor, [1116](#)
- QuantLib::TimeBasket, [1117](#)
- QuantLib::TimeGrid, [1118](#)
- QuantLib::TimeGrid
  - TimeGrid, [1119](#)
- QuantLib::TimeSeries, [1120](#)
- QuantLib::TimeSeries
  - TimeSeries, [1121](#)
- QuantLib::TqrEigenDecomposition, [1122](#)
- QuantLib::TransformedGrid, [1123](#)
- QuantLib::TrapezoidIntegral, [1124](#)
- QuantLib::Tree, [1126](#)
- QuantLib::TreeCapFloorEngine, [1127](#)
- QuantLib::TreeSwaptionEngine, [1128](#)
- QuantLib::TreeVanillaSwapEngine, [1129](#)
- QuantLib::TridiagonalOperator, [1130](#)
- QuantLib::TridiagonalOperator::TimeSetter, [1132](#)
- QuantLib::Trigeorgis, [1133](#)
- QuantLib::TrinomialTree, [1134](#)
- QuantLib::TRLCurrency, [1135](#)
- QuantLib::TRLibor, [1136](#)
- QuantLib::TRYCurrency, [1137](#)
- QuantLib::TsiveriotisFernandesLattice, [1138](#)
- QuantLib::TsiveriotisFernandesLattice
  - partialRollback, [1138](#)
  - rollback, [1138](#)
- QuantLib::TTDCurrency, [1140](#)
- QuantLib::Turkey, [1141](#)
- QuantLib::TWDCurrency, [1142](#)
- QuantLib::TwoFactorModel, [1143](#)
- QuantLib::TwoFactorModel::ShortRateDynamics, [1144](#)
- QuantLib::TwoFactorModel::ShortRateTree, [1145](#)
- QuantLib::TypePayoff, [1146](#)
- QuantLib::Ukraine, [1147](#)
  - USE, [1148](#)
- QuantLib::Ukraine
  - Market, [1148](#)
- QuantLib::UnitedKingdom, [1149](#)
  - Exchange, [1150](#)
  - Settlement, [1150](#)
- QuantLib::UnitedKingdom
  - Market, [1150](#)
- QuantLib::UnitedStates, [1151](#)
  - GovernmentBond, [1153](#)
  - NERC, [1153](#)
  - NYSE, [1153](#)

- Settlement, [1153](#)
- QuantLib::UnitedStates
  - Market, [1153](#)
- QuantLib::UpFrontIndexedCoupon, [1154](#)
- QuantLib::UpRounding, [1155](#)
- QuantLib::USDCurrency, [1156](#)
- QuantLib::USDLibor, [1157](#)
- QuantLib::Value, [1158](#)
- QuantLib::VanillaCMSCouponPricer, [1159](#)
- QuantLib::VanillaOption, [1160](#)
- QuantLib::VanillaOption::engine, [1161](#)
- QuantLib::VanillaSwap, [1162](#)
- QuantLib::VanillaSwap
  - fetchResults, [1163](#)
  - setupArguments, [1163](#)
  - VanillaSwap, [1163](#)
- QuantLib::VanillaSwap::arguments, [1164](#)
- QuantLib::VanillaSwap::results, [1165](#)
- QuantLib::VarianceSwap, [1166](#)
- QuantLib::VarianceSwap
  - fetchResults, [1167](#)
  - performCalculations, [1167](#)
  - setupArguments, [1167](#)
  - setupExpired, [1167](#)
- QuantLib::VarianceSwap::arguments, [1169](#)
- QuantLib::VarianceSwap::engine, [1170](#)
- QuantLib::VarianceSwap::results, [1171](#)
- QuantLib::Vasicek, [1172](#)
- QuantLib::Vasicek::Dynamics, [1174](#)
- QuantLib::VEBCurrency, [1175](#)
- QuantLib::Visitor, [1176](#)
- QuantLib::Xibor, [1177](#)
- QuantLib::Xibor
  - frequency, [1178](#)
- QuantLib::YieldTermStructure, [1179](#)
- QuantLib::YieldTermStructure
  - discount, [1181](#)
  - forwardRate, [1181](#)
  - parRate, [1181](#)
  - YieldTermStructure, [1180](#)
  - zeroRate, [1181](#)
- QuantLib::ZARCurrency, [1183](#)
- QuantLib::ZeroCondition, [1184](#)
- QuantLib::ZeroCouponBond, [1185](#)
- QuantLib::ZeroCouponBond
  - ZeroCouponBond, [1185](#)
- QuantLib::ZeroSpreadedTermStructure, [1186](#)
- QuantLib::ZeroYield, [1188](#)
- QuantLib::ZeroYieldStructure, [1189](#)
- QuantLib::ZeroYieldStructure
  - discountImpl, [1189](#)
- QuantLib::Zibor, [1190](#)
- Quanto option engines, [121](#)
- Quarterly
  - datetime, [109](#)
- rankReducedSqrt
  - QuantLib::Matrix, [860](#)
- recalculate
  - QuantLib::LazyObject, [799](#)
- regret
  - QuantLib::GenericRiskStatistics, [693](#)
- removeHoliday
  - QuantLib::Calendar, [278](#)
- reset
  - QuantLib::DiscretizedAsset, [408](#)
  - QuantLib::DiscretizedDiscountBond, [410](#)
  - QuantLib::DiscretizedOption, [411](#)
- rollback
  - QuantLib::FiniteDifferenceModel, [600](#)
  - QuantLib::Lattice, [794](#)
  - QuantLib::NumericalMethod, [934](#)
  - QuantLib::TsiveriotisFernandesLattice, [1138](#)
- Rounding
  - QuantLib::Rounding, [1011](#)
- Schedule
  - QuantLib::Schedule, [1020](#)
- SecondDerivative
  - QuantLib::CubicSpline, [374](#)
- secondDerivativeAtCenter
  - QuantLib::SampledCurve, [1016](#)
- Semiannual
  - datetime, [109](#)
- semiDeviation
  - QuantLib::GenericRiskStatistics, [693](#)
- semiVariance
  - QuantLib::GenericRiskStatistics, [693](#)
- sequencestatistics.hpp
  - DEFINE\_SEQUENCE\_STAT\_CONST\_-METHOD\_DOUBLE, [1432](#)
  - DEFINE\_SEQUENCE\_STAT\_CONST\_-METHOD\_VOID, [1431](#)
- setMaxEvaluations
  - QuantLib::Solver1D, [1055](#)
- setPricingEngine
  - QuantLib::Instrument, [747](#)
- setTermStructure
  - QuantLib::FixedCouponBondHelper, [608](#)
  - QuantLib::RateHelper, [1003](#)
  - QuantLib::SwapRateHelper, [1084](#)
- setTime
  - QuantLib::BoundaryCondition, [260](#)
  - QuantLib::DirichletBC, [399](#)
  - QuantLib::NeumannBC, [919](#)
- Settlement
  - QuantLib::Brazil, [264](#)

- QuantLib::Germany, 700
- QuantLib::Italy, 777
- QuantLib::UnitedKingdom, 1150
- QuantLib::UnitedStates, 1153
- settlementDate
  - QuantLib::ForwardRateAgreement, 635
- setupArguments
  - QuantLib::AssetSwap, 193
  - QuantLib::BarrierOption, 206
  - QuantLib::BasketOption, 210
  - QuantLib::CapFloor, 292
  - QuantLib::CliquetOption, 318
  - QuantLib::ContinuousAveragingAsian-Option, 341
  - QuantLib::ContinuousFixedLookback-Option, 345
  - QuantLib::ContinuousFloatingLookback-Option, 349
  - QuantLib::DiscreteAveragingAsian-Option, 403
  - QuantLib::DividendVanillaOption, 417
  - QuantLib::ForwardVanillaOption, 645
  - QuantLib::Instrument, 747
  - QuantLib::MultiAssetOption, 906
  - QuantLib::OneAssetOption, 944
  - QuantLib::OneAssetStrikedOption, 948
  - QuantLib::QuantoForwardVanillaOption, 991
  - QuantLib::QuantoVanillaOption, 997
  - QuantLib::Swaption, 1086
  - QuantLib::VanillaSwap, 1163
  - QuantLib::VarianceSwap, 1167
- setupExpired
  - QuantLib::Instrument, 748
  - QuantLib::MultiAssetOption, 906
  - QuantLib::OneAssetOption, 945
  - QuantLib::OneAssetStrikedOption, 949
  - QuantLib::QuantoVanillaOption, 997
  - QuantLib::Swap, 1080
  - QuantLib::VarianceSwap, 1167
- SGX
  - QuantLib::Singapore, 1040
- Short-rate modelling framework, 128
- shortfall
  - QuantLib::GenericRiskStatistics, 694
- Side
  - QuantLib::BoundaryCondition, 259
- Simplex
  - QuantLib::Simplex, 1037
- skewness
  - QuantLib::GeneralStatistics, 684
  - QuantLib::IncrementalStatistics, 736
- skipTo
  - QuantLib::SobolRsg, 1052
- SobolRsg
  - QuantLib::SobolRsg, 1052
- solve
  - QuantLib::Solver1D, 1055
- spotIncome
  - QuantLib::FixedCouponBondForward, 606
  - QuantLib::ForwardRateAgreement, 636
- spotValue
  - QuantLib::ForwardRateAgreement, 636
- standardDeviation
  - QuantLib::GeneralStatistics, 684
  - QuantLib::IncrementalStatistics, 736
- standardDeviations
  - QuantLib::CovarianceDecomposition, 365
- stdDeviation
  - QuantLib::G2ForwardProcess, 655
  - QuantLib::G2Process, 657
  - QuantLib::HullWhiteForwardProcess, 719
  - QuantLib::HullWhiteProcess, 721
  - QuantLib::OrnsteinUhlenbeckProcess, 960
  - QuantLib::StochasticProcess, 1066
  - QuantLib::StochasticProcess1D, 1069
  - QuantLib::StochasticProcessArray, 1073
- Stochastic processes, 142
- Swap
  - QuantLib::Swap, 1080
- Swaption engines, 122
- SwaptionVolatilityStructure
  - QuantLib::SwaptionVolatilityStructure, 1101
- SymmetricSchurDecomposition
  - QuantLib::SymmetricSchur-Decomposition, 1104
- targetValueAndGradient
  - QuantLib::LeastSquareProblem, 801
- Template capabilities, 151
- templateMacros
  - QL\_TYPENAME, 151
- Term structures, 144
- TermStructure
  - QuantLib::TermStructure, 1110
- time
  - QuantLib::GeneralizedBlackScholes-Process, 682
  - QuantLib::HestonProcess, 709
  - QuantLib::Merton76Process, 892
  - QuantLib::StochasticProcess, 1067
  - QuantLib::StochasticProcessArray, 1073
- TimeGrid
  - QuantLib::TimeGrid, 1119
- TimeSeries
  - QuantLib::TimeSeries, 1121

- topPercentile
  - QuantLib::GeneralStatistics, 685
- TSEC
  - QuantLib::Taiwan, 1107
- Type
  - QuantLib::ExchangeRate, 576
  - QuantLib::Rounding, 1010
- Unadjusted
  - datetime, 109
- UnadjustedMonthEnd
  - datetime, 109
- underlyingArgs
  - QuantLib::QuantoEngine, 989
- underlyingIncome\_
  - QuantLib::Forward, 625
- underlyingSpotValue\_
  - QuantLib::Forward, 625
- underlyingSwap
  - QuantLib::SwapIndex, 1082
- unfreeze
  - QuantLib::LazyObject, 799
- Up
  - QuantLib::Rounding, 1011
- update
  - QuantLib::BlackCapFloorEngine, 232
  - QuantLib::BlackSwaptionEngine, 243
  - QuantLib::CalibratedModel, 284
  - QuantLib::CalibrationHelper, 286
  - QuantLib::CapVolatilityVector, 304
  - QuantLib::CompositeQuote, 332
  - QuantLib::DerivedQuote, 398
  - QuantLib::ExtendedDiscountCurve, 588
  - QuantLib::FlatForward, 614
  - QuantLib::FloatingRateCoupon, 619
  - QuantLib::GeneralizedBlackScholes-  
Process, 682
  - QuantLib::GenericModelEngine, 691
  - QuantLib::InterestRateIndex, 754
  - QuantLib::LatticeShortRateModelEngine,  
797
  - QuantLib::LazyObject, 798
  - QuantLib::MCHullWhiteCapFloorEngine,  
878
  - QuantLib::Observer, 942
  - QuantLib::PiecewiseYieldCurve, 973
  - QuantLib::PiecewiseZeroSpreadedTerm-  
Structure, 975
  - QuantLib::RateHelper, 1003
  - QuantLib::RelativeDateRateHelper, 1004
  - QuantLib::StochasticProcess, 1067
  - QuantLib::TermStructure, 1110
- USE
  - QuantLib::Ukraine, 1148
- Utilities, 146
- valueAtCenter
  - QuantLib::SampledCurve, 1016
- valueAtRisk
  - QuantLib::GenericRiskStatistics, 693
- valueDate\_
  - QuantLib::Forward, 625
- Vanilla option engines, 123
- vanillaengines
  - FDAmericanEngine, 124
  - FDDividendAmericanEngine, 124
  - FDDividendEuropeanEngine, 125
  - FDDividendShoutEngine, 125
  - FDShoutEngine, 125
- VanillaSwap
  - QuantLib::VanillaSwap, 1163
- variance
  - QuantLib::Abcd, 159
  - QuantLib::EulerDiscretization, 442
  - QuantLib::GeneralStatistics, 684
  - QuantLib::HullWhiteForwardProcess, 719
  - QuantLib::HullWhiteProcess, 721
  - QuantLib::IncrementalStatistics, 736
  - QuantLib::OrnsteinUhlenbeckProcess, 961
  - QuantLib::StochasticProcess1D, 1069
- variances
  - QuantLib::CovarianceDecomposition, 365
- volatility
  - QuantLib::Abcd, 158
- Weekday
  - datetime, 109
- Weekly
  - datetime, 109
- Xetra
  - QuantLib::Germany, 700
- yield
  - QuantLib::Bond, 257, 258
- YieldTermStructure
  - QuantLib::YieldTermStructure, 1180
- yieldtermstructures
  - DiscountCurve, 145
- ZeroCouponBond
  - QuantLib::ZeroCouponBond, 1185
- zeroRate
  - QuantLib::YieldTermStructure, 1181
- zeroYieldImpl
  - QuantLib::CompoundForward, 334
  - QuantLib::ExtendedDiscountCurve, 588
  - QuantLib::ForwardRateStructure, 639
  - QuantLib::InterpolatedForwardCurve, 758