

# QuantLib

**An open source library for quantitative finance**

Version 0.3.12

Generated by Doxygen 1.4.6

20 Mar 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 . . . . .	41
1.10	The QuantLib Group . . . . .	42
1.11	QuantLib License . . . . .	43
<b>2</b>	<b>QuantLib Module Index</b>	<b>45</b>
2.1	QuantLib Modules . . . . .	45
<b>3</b>	<b>QuantLib Hierarchical Index</b>	<b>47</b>
3.1	QuantLib Class Hierarchy . . . . .	47
<b>4</b>	<b>QuantLib Class Index</b>	<b>63</b>
4.1	QuantLib Class List . . . . .	63
<b>5</b>	<b>QuantLib File Index</b>	<b>77</b>
5.1	QuantLib File List . . . . .	77
<b>6</b>	<b>QuantLib Module Documentation</b>	<b>87</b>
6.1	Numeric types . . . . .	87
6.2	Currencies and FX rates . . . . .	89
6.3	Date and time calculations . . . . .	93
6.4	Calendars . . . . .	96

6.5	Day counters . . . . .	99
6.6	Pricing engines . . . . .	100
6.7	Asian option engines . . . . .	101
6.8	Barrier option engines . . . . .	102
6.9	Basket option engines . . . . .	103
6.10	Cap/floor engines . . . . .	104
6.11	Cliquet option engines . . . . .	105
6.12	Forward option engines . . . . .	106
6.13	Quanto option engines . . . . .	107
6.14	Swaption engines . . . . .	108
6.15	Vanilla option engines . . . . .	109
6.16	Finite-differences framework . . . . .	112
6.17	Short-rate modelling framework . . . . .	119
6.18	Financial instruments . . . . .	122
6.19	Lattice methods . . . . .	125
6.20	Math tools . . . . .	128
6.21	Monte Carlo framework . . . . .	130
6.22	Design patterns . . . . .	136
6.23	Term structures . . . . .	137
6.24	Utilities . . . . .	139
6.25	QuantLib macros . . . . .	141
6.26	Generic macros . . . . .	142
6.27	Numeric limits . . . . .	143
6.28	Template capabilities . . . . .	144
6.29	Iterator support . . . . .	145
6.30	Output manipulators . . . . .	146
6.31	Debugging macros . . . . .	147
<b>7</b>	<b>QuantLib Class Documentation</b>	<b>151</b>
7.1	Actual360 Class Reference . . . . .	151
7.2	Actual365Fixed Class Reference . . . . .	152
7.3	ActualActual Class Reference . . . . .	153
7.4	AcyclicVisitor Class Reference . . . . .	154
7.5	AdditiveEQPBinoomialTree Class Reference . . . . .	155
7.6	AffineModel Class Reference . . . . .	156
7.7	AffineTermStructure Class Reference . . . . .	157
7.8	AmericanCondition Class Reference . . . . .	159



7.9	AmericanExercise Class Reference . . . . .	160
7.10	AmericanPayoffAtExpiry Class Reference . . . . .	161
7.11	AmericanPayoffAtHit Class Reference . . . . .	162
7.12	AnalyticBarrierEngine Class Reference . . . . .	163
7.13	AnalyticCapFloorEngine Class Reference . . . . .	164
7.14	AnalyticCliquetEngine Class Reference . . . . .	165
7.15	AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference . . . . .	166
7.16	AnalyticDigitalAmericanEngine Class Reference . . . . .	167
7.17	AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference . . . . .	168
7.18	AnalyticDividendEuropeanEngine Class Reference . . . . .	169
7.19	AnalyticEuropeanEngine Class Reference . . . . .	170
7.20	AnalyticHestonEngine Class Reference . . . . .	171
7.21	AnalyticPerformanceEngine Class Reference . . . . .	172
7.22	Argentina Class Reference . . . . .	173
7.23	Arguments Class Reference . . . . .	175
7.24	ArmijoLineSearch Class Reference . . . . .	176
7.25	Array Class Reference . . . . .	177
7.26	ARSCurrency Class Reference . . . . .	180
7.27	AssetOrNothingPayoff Class Reference . . . . .	181
7.28	ATSCurrency Class Reference . . . . .	182
7.29	AUDCurrency Class Reference . . . . .	183
7.30	AUDLibor Class Reference . . . . .	184
7.31	Australia Class Reference . . . . .	185
7.32	Average Struct Reference . . . . .	186
7.33	BackwardFlat Class Reference . . . . .	187
7.34	BackwardFlatInterpolation Class Reference . . . . .	188
7.35	BaroneAdesiWhaleyApproximationEngine Class Reference . . . . .	189
7.36	Barrier Struct Reference . . . . .	190
7.37	BarrierOption Class Reference . . . . .	191
7.38	BarrierOption::arguments Class Reference . . . . .	193
7.39	BarrierOption::engine Class Reference . . . . .	194
7.40	BasketOption Class Reference . . . . .	195
7.41	BasketOption::arguments Class Reference . . . . .	197
7.42	BasketOption::engine Class Reference . . . . .	198
7.43	BatesEngine Class Reference . . . . .	199
7.44	BatesModel Class Reference . . . . .	201

7.45	BDTCurrency Class Reference . . . . .	202
7.46	BEFCurrency Class Reference . . . . .	203
7.47	BermudanExercise Class Reference . . . . .	204
7.48	BGLCurrency Class Reference . . . . .	205
7.49	Bicubic Class Reference . . . . .	206
7.50	BicubicSpline Class Reference . . . . .	207
7.51	Bilinear Class Reference . . . . .	208
7.52	BilinearInterpolation Class Reference . . . . .	209
7.53	BinomialConvertibleEngine Class Template Reference . . . . .	210
7.54	BinomialDistribution Class Reference . . . . .	211
7.55	BinomialTree Class Template Reference . . . . .	212
7.56	BinomialVanillaEngine Class Template Reference . . . . .	213
7.57	Bisection Class Reference . . . . .	214
7.58	BivariateCumulativeNormalDistributionDr78 Class Reference . . . . .	215
7.59	BivariateCumulativeNormalDistributionWe04DP Class Reference . . . . .	216
7.60	BjerkstrandStenslandApproximationEngine Class Reference . . . . .	217
7.61	BlackCapFloorEngine Class Reference . . . . .	218
7.62	BlackConstantVol Class Reference . . . . .	219
7.63	BlackFormula Class Reference . . . . .	221
7.64	BlackKarasinski Class Reference . . . . .	223
7.65	BlackKarasinski::Dynamics Class Reference . . . . .	224
7.66	BlackModel Class Reference . . . . .	225
7.67	BlackScholesLattice Class Template Reference . . . . .	227
7.68	BlackScholesProcess Class Reference . . . . .	228
7.69	BlackSwaptionEngine Class Reference . . . . .	230
7.70	BlackVarianceCurve Class Reference . . . . .	231
7.71	BlackVarianceSurface Class Reference . . . . .	233
7.72	BlackVarianceTermStructure Class Reference . . . . .	235
7.73	BlackVolatilityTermStructure Class Reference . . . . .	237
7.74	BlackVolTermStructure Class Reference . . . . .	239
7.75	Bond Class Reference . . . . .	242
7.76	BoundaryCondition Class Template Reference . . . . .	246
7.77	BoundaryConstraint Class Reference . . . . .	248
7.78	BoxMullerGaussianRng Class Template Reference . . . . .	249
7.79	BPSBasketCalculator Class Reference . . . . .	250
7.80	Brazil Class Reference . . . . .	251

7.81	Brent Class Reference . . . . .	252
7.82	Bridge Class Template Reference . . . . .	253
7.83	BRLCurrency Class Reference . . . . .	254
7.84	BrownianBridge Class Template Reference . . . . .	255
7.85	BSMOperator Class Reference . . . . .	256
7.86	BYRCurrency Class Reference . . . . .	257
7.87	CADCurrency Class Reference . . . . .	258
7.88	CADLibor Class Reference . . . . .	259
7.89	Calendar Class Reference . . . . .	260
7.90	Calendar::OrthodoxImpl Class Reference . . . . .	265
7.91	Calendar::WesternImpl Class Reference . . . . .	266
7.92	CalendarImpl Class Reference . . . . .	267
7.93	CalibrationHelper Class Reference . . . . .	268
7.94	Canada Class Reference . . . . .	270
7.95	Cap Class Reference . . . . .	271
7.96	CapFloor Class Reference . . . . .	272
7.97	CapFloor::arguments Class Reference . . . . .	274
7.98	CapFloor::results Class Reference . . . . .	275
7.99	CapHelper Class Reference . . . . .	276
7.100	CapletConstantVolatility Class Reference . . . . .	277
7.101	CapletVolatilityStructure Class Reference . . . . .	279
7.102	CapVolatilityStructure Class Reference . . . . .	281
7.103	CapVolatilityVector Class Reference . . . . .	283
7.104	CashFlow Class Reference . . . . .	285
7.105	Cashflows Class Reference . . . . .	287
7.106	CashOrNothingPayoff Class Reference . . . . .	290
7.107	Cdor Class Reference . . . . .	291
7.108	CeilingTruncation Class Reference . . . . .	292
7.109	CHFCurrency Class Reference . . . . .	293
7.110	CHFLibor Class Reference . . . . .	294
7.111	China Class Reference . . . . .	295
7.112	CLGaussianRng Class Template Reference . . . . .	296
7.113	CliquetOption Class Reference . . . . .	297
7.114	CliquetOption::arguments Class Reference . . . . .	299
7.115	CliquetOption::engine Class Reference . . . . .	300
7.116	ClosestRounding Class Reference . . . . .	301

7.117	CLPCurrency Class Reference . . . . .	302
7.118	CNYCurrency Class Reference . . . . .	303
7.119	Collar Class Reference . . . . .	304
7.120	Composite Class Template Reference . . . . .	305
7.121	CompositeConstraint Class Reference . . . . .	306
7.122	CompositeQuote Class Template Reference . . . . .	307
7.123	CompoundForward Class Reference . . . . .	308
7.124	ConjugateGradient Class Reference . . . . .	310
7.125	ConstantParameter Class Reference . . . . .	311
7.126	Constraint Class Reference . . . . .	312
7.127	ConstraintImpl Class Reference . . . . .	313
7.128	ContinuousAveragingAsianOption Class Reference . . . . .	314
7.129	ContinuousAveragingAsianOption::arguments Class Reference . . . . .	316
7.130	ContinuousAveragingAsianOption::engine Class Reference . . . . .	317
7.131	ConvergenceStatistics Class Template Reference . . . . .	318
7.132	ConvertibleBond::option::arguments Class Reference . . . . .	319
7.133	ConvertibleBond::option::engine Class Reference . . . . .	320
7.134	ConvertibleFixedCouponBond Class Reference . . . . .	321
7.135	ConvertibleFloatingRateBond Class Reference . . . . .	322
7.136	ConvertibleZeroCouponBond Class Reference . . . . .	323
7.137	COPCurrency Class Reference . . . . .	324
7.138	CostFunction Class Reference . . . . .	325
7.139	Coupon Class Reference . . . . .	326
7.140	CovarianceDecomposition Class Reference . . . . .	328
7.141	CoxIngersollRoss Class Reference . . . . .	329
7.142	CoxIngersollRoss::Dynamics Class Reference . . . . .	331
7.143	CoxRossRubinstein Class Reference . . . . .	332
7.144	CrankNicolson Class Template Reference . . . . .	333
7.145	Cubic Class Reference . . . . .	335
7.146	CubicSpline Class Reference . . . . .	336
7.147	CumulativeBinomialDistribution Class Reference . . . . .	338
7.148	CumulativeNormalDistribution Class Reference . . . . .	339
7.149	CumulativePoissonDistribution Class Reference . . . . .	340
7.150	CuriouslyRecurringTemplate Class Template Reference . . . . .	341
7.151	Currency Class Reference . . . . .	342
7.152	CYPCurrency Class Reference . . . . .	346

7.153	CzechRepublic Class Reference . . . . .	347
7.154	CZKCurrency Class Reference . . . . .	349
7.155	Date Class Reference . . . . .	350
7.156	DayCounter Class Reference . . . . .	354
7.157	DayCounterImpl Class Reference . . . . .	356
7.158	DEMCurrency Class Reference . . . . .	357
7.159	Denmark Class Reference . . . . .	358
7.160	DepositRateHelper Class Reference . . . . .	359
7.161	DerivedQuote Class Template Reference . . . . .	361
7.162	DirichletBC Class Reference . . . . .	362
7.163	Discount Struct Reference . . . . .	364
7.164	DiscrepancyStatistics Class Reference . . . . .	365
7.165	DiscreteAveragingAsianOption Class Reference . . . . .	366
7.166	DiscreteAveragingAsianOption::arguments Class Reference . . . . .	368
7.167	DiscreteAveragingAsianOption::engine Class Reference . . . . .	369
7.168	DiscreteGeometricASO Class Reference . . . . .	370
7.169	DiscretizedAsset Class Reference . . . . .	371
7.170	DiscretizedDiscountBond Class Reference . . . . .	374
7.171	DiscretizedOption Class Reference . . . . .	375
7.172	Disposable Class Template Reference . . . . .	377
7.173	Dividend Class Reference . . . . .	378
7.174	DividendVanillaOption Class Reference . . . . .	380
7.175	DividendVanillaOption::arguments Class Reference . . . . .	382
7.176	DividendVanillaOption::engine Class Reference . . . . .	383
7.177	DKKCurrency Class Reference . . . . .	384
7.178	DKKLibor Class Reference . . . . .	385
7.179	DMinus Class Reference . . . . .	386
7.180	DownRounding Class Reference . . . . .	387
7.181	DPlus Class Reference . . . . .	388
7.182	DPlusDMinus Class Reference . . . . .	389
7.183	DriftTermStructure Class Reference . . . . .	390
7.184	Duration Struct Reference . . . . .	392
7.185	DZero Class Reference . . . . .	393
7.186	EarlyExercise Class Reference . . . . .	394
7.187	EEKCurrency Class Reference . . . . .	395
7.188	EndCriteria Class Reference . . . . .	396

7.189	EqualJumpsBinomialTree Class Template Reference . . . . .	398
7.190	EqualProbabilitiesBinomialTree Class Template Reference . . . . .	399
7.191	Error Class Reference . . . . .	400
7.192	ErrorFunction Class Reference . . . . .	401
7.193	ESPCurrency Class Reference . . . . .	402
7.194	EulerDiscretization Class Reference . . . . .	403
7.195	EURCurrency Class Reference . . . . .	405
7.196	Euribor Class Reference . . . . .	406
7.197	EURLibor Class Reference . . . . .	407
7.198	EuropeanExercise Class Reference . . . . .	408
7.199	EuropeanOption Class Reference . . . . .	409
7.200	Event Class Reference . . . . .	410
7.201	ExchangeRate Class Reference . . . . .	412
7.202	ExchangeRateManager Class Reference . . . . .	414
7.203	Exercise Class Reference . . . . .	416
7.204	ExplicitEuler Class Template Reference . . . . .	417
7.205	ExtendedCoxIngersollRoss Class Reference . . . . .	419
7.206	ExtendedCoxIngersollRoss::Dynamics Class Reference . . . . .	421
7.207	ExtendedCoxIngersollRoss::FittingParameter Class Reference . . . . .	422
7.208	ExtendedDiscountCurve Class Reference . . . . .	423
7.209	Extrapolator Class Reference . . . . .	425
7.210	Factorial Class Reference . . . . .	426
7.211	FalsePosition Class Reference . . . . .	427
7.212	FaureRsg Class Reference . . . . .	428
7.213	FDAmericanCondition Class Template Reference . . . . .	429
7.214	FDBermudanEngine Class Reference . . . . .	430
7.215	FDDividendEngineMerton73 Class Reference . . . . .	431
7.216	FDDividendEngineShiftScale Class Reference . . . . .	432
7.217	FDEuropeanEngine Class Reference . . . . .	433
7.218	FDStepConditionEngine Class Reference . . . . .	434
7.219	FIMCurrency Class Reference . . . . .	435
7.220	FiniteDifferenceModel Class Template Reference . . . . .	436
7.221	Finland Class Reference . . . . .	437
7.222	FixedCouponBond Class Reference . . . . .	438
7.223	FixedCouponBondHelper Class Reference . . . . .	440
7.224	FixedDividend Class Reference . . . . .	442

7.225	FixedRateCoupon Class Reference . . . . .	444
7.226	FlatForward Class Reference . . . . .	446
7.227	FloatingRateBond Class Reference . . . . .	448
7.228	FloatingRateCoupon Class Reference . . . . .	450
7.229	Floor Class Reference . . . . .	452
7.230	FloorTruncation Class Reference . . . . .	453
7.231	ForwardEngine Class Template Reference . . . . .	454
7.232	ForwardFlat Class Reference . . . . .	455
7.233	ForwardFlatInterpolation Class Reference . . . . .	456
7.234	ForwardOptionArguments Class Template Reference . . . . .	457
7.235	ForwardPerformanceEngine Class Template Reference . . . . .	458
7.236	ForwardRate Struct Reference . . . . .	459
7.237	ForwardRateStructure Class Reference . . . . .	460
7.238	ForwardSpreadedTermStructure Class Reference . . . . .	462
7.239	ForwardVanillaOption Class Reference . . . . .	464
7.240	FractionalDividend Class Reference . . . . .	466
7.241	FraRateHelper Class Reference . . . . .	468
7.242	FRFCurrency Class Reference . . . . .	470
7.243	FuturesRateHelper Class Reference . . . . .	471
7.244	G2 Class Reference . . . . .	472
7.245	G2::FittingParameter Class Reference . . . . .	474
7.246	G2SwaptionEngine Class Reference . . . . .	475
7.247	GammaFunction Class Reference . . . . .	476
7.248	GapPayoff Class Reference . . . . .	477
7.249	GaussChebyshev2thIntegration Class Reference . . . . .	478
7.250	GaussChebyshevIntegration Class Reference . . . . .	479
7.251	GaussGegenbauerIntegration Class Reference . . . . .	480
7.252	GaussHermiteIntegration Class Reference . . . . .	481
7.253	GaussHermitePolynomial Class Reference . . . . .	482
7.254	GaussHyperbolicIntegration Class Reference . . . . .	483
7.255	GaussHyperbolicPolynomial Class Reference . . . . .	484
7.256	GaussianOrthogonalPolynomial Class Reference . . . . .	485
7.257	GaussianQuadrature Class Reference . . . . .	486
7.258	GaussianStatistics Class Template Reference . . . . .	487
7.259	GaussJacobiIntegration Class Reference . . . . .	490
7.260	GaussJacobiPolynomial Class Reference . . . . .	491

7.261	GaussLaguerreIntegration Class Reference . . . . .	492
7.262	GaussLaguerrePolynomial Class Reference . . . . .	493
7.263	GaussLegendreIntegration Class Reference . . . . .	494
7.264	GBPCurrency Class Reference . . . . .	495
7.265	GBPLibor Class Reference . . . . .	496
7.266	GeneralStatistics Class Reference . . . . .	497
7.267	GenericEngine Class Template Reference . . . . .	500
7.268	GenericModelEngine Class Template Reference . . . . .	501
7.269	GenericRiskStatistics Class Template Reference . . . . .	502
7.270	GeometricBrownianMotionProcess Class Reference . . . . .	505
7.271	Germany Class Reference . . . . .	506
7.272	GRDCurrency Class Reference . . . . .	509
7.273	Greeks Class Reference . . . . .	510
7.274	HaltonRsg Class Reference . . . . .	511
7.275	Handle Class Template Reference . . . . .	512
7.276	HestonModel Class Reference . . . . .	513
7.277	HestonModelHelper Class Reference . . . . .	514
7.278	HestonProcess Class Reference . . . . .	515
7.279	History Class Reference . . . . .	517
7.280	History::const_iterator Class Reference . . . . .	520
7.281	History::Entry Class Reference . . . . .	521
7.282	HKDCurrency Class Reference . . . . .	522
7.283	HongKong Class Reference . . . . .	523
7.284	HUFCurrency Class Reference . . . . .	525
7.285	HullWhite Class Reference . . . . .	526
7.286	HullWhite::Dynamics Class Reference . . . . .	528
7.287	HullWhite::FittingParameter Class Reference . . . . .	529
7.288	Hungary Class Reference . . . . .	530
7.289	Iceland Class Reference . . . . .	531
7.290	IEPCurrency Class Reference . . . . .	533
7.291	ILSCurrency Class Reference . . . . .	534
7.292	IMM Struct Reference . . . . .	535
7.293	ImplicitEuler Class Template Reference . . . . .	536
7.294	ImpliedTermStructure Class Reference . . . . .	537
7.295	ImpliedVolTermStructure Class Reference . . . . .	539
7.296	InArrearIndexedCoupon Class Reference . . . . .	541



7.297	IncrementalStatistics Class Reference . . . . .	543
7.298	Index Class Reference . . . . .	546
7.299	IndexedCoupon Class Reference . . . . .	547
7.300	IndexManager Class Reference . . . . .	549
7.301	India Class Reference . . . . .	550
7.302	Indonesia Class Reference . . . . .	552
7.303	INRCurrency Class Reference . . . . .	554
7.304	Instrument Class Reference . . . . .	555
7.305	IntegralEngine Class Reference . . . . .	558
7.306	InterestRate Class Reference . . . . .	559
7.307	InterpolatedDiscountCurve Class Template Reference . . . . .	562
7.308	InterpolatedForwardCurve Class Template Reference . . . . .	564
7.309	InterpolatedZeroCurve Class Template Reference . . . . .	566
7.310	Interpolation Class Reference . . . . .	568
7.311	Interpolation2D Class Reference . . . . .	569
7.312	Interpolation2D::templateImpl Class Template Reference . . . . .	570
7.313	Interpolation2DImpl Class Reference . . . . .	571
7.314	Interpolation::templateImpl Class Template Reference . . . . .	572
7.315	InterpolationImpl Class Reference . . . . .	573
7.316	InverseCumulativeNormal Class Reference . . . . .	574
7.317	InverseCumulativePoisson Class Reference . . . . .	575
7.318	InverseCumulativeRng Class Template Reference . . . . .	576
7.319	InverseCumulativeRsg Class Template Reference . . . . .	577
7.320	IQDCurrency Class Reference . . . . .	578
7.321	IRRCurrency Class Reference . . . . .	579
7.322	ISKCurrency Class Reference . . . . .	580
7.323	Italy Class Reference . . . . .	581
7.324	ITLCurrency Class Reference . . . . .	583
7.325	JamshidianSwaptionEngine Class Reference . . . . .	584
7.326	Japan Class Reference . . . . .	585
7.327	JarrowRudd Class Reference . . . . .	587
7.328	Jibar Class Reference . . . . .	588
7.329	JointCalendar Class Reference . . . . .	589
7.330	JPYCurrency Class Reference . . . . .	590
7.331	JPYLibor Class Reference . . . . .	591
7.332	JumpDiffusionEngine Class Reference . . . . .	592

7.333	JuQuadraticApproximationEngine Class Reference . . . . .	593
7.334	KnuthUniformRng Class Reference . . . . .	594
7.335	KronrodIntegral Class Reference . . . . .	595
7.336	KRWCurrency Class Reference . . . . .	596
7.337	KWDCurrency Class Reference . . . . .	597
7.338	Lattice Class Template Reference . . . . .	598
7.339	Lattice1D Class Template Reference . . . . .	600
7.340	Lattice2D Class Template Reference . . . . .	601
7.341	LatticeShortRateModelEngine Class Template Reference . . . . .	602
7.342	LazyObject Class Reference . . . . .	603
7.343	LeastSquareFunction Class Reference . . . . .	605
7.344	LeastSquareProblem Class Reference . . . . .	606
7.345	LecuyerUniformRng Class Reference . . . . .	607
7.346	LeisenReimer Class Reference . . . . .	608
7.347	LevenbergMarquardt Class Reference . . . . .	609
7.348	LexicographicalView Class Template Reference . . . . .	610
7.349	LfmCovarianceParameterization Class Reference . . . . .	612
7.350	LfmCovarianceProxy Class Reference . . . . .	613
7.351	LfmHullWhiteParameterization Class Reference . . . . .	614
7.352	LfmSwaptionEngine Class Reference . . . . .	615
7.353	Libor Class Reference . . . . .	616
7.354	LiborForwardModel Class Reference . . . . .	617
7.355	LiborForwardModelProcess Class Reference . . . . .	619
7.356	Linear Class Reference . . . . .	621
7.357	LinearInterpolation Class Reference . . . . .	622
7.358	LineSearch Class Reference . . . . .	623
7.359	Link Class Template Reference . . . . .	625
7.360	LmCorrelationModel Class Reference . . . . .	627
7.361	LmExponentialCorrelationModel Class Reference . . . . .	628
7.362	LmLinearExponentialVolatilityModel Class Reference . . . . .	629
7.363	LmVolatilityModel Class Reference . . . . .	630
7.364	LocalConstantVol Class Reference . . . . .	631
7.365	LocalVolCurve Class Reference . . . . .	632
7.366	LocalVolSurface Class Reference . . . . .	634
7.367	LocalVolTermStructure Class Reference . . . . .	636
7.368	LogLinear Class Reference . . . . .	638

7.369	LogLinearInterpolation Class Reference . . . . .	639
7.370	LTLCurrency Class Reference . . . . .	640
7.371	LUFCurrency Class Reference . . . . .	641
7.372	LVLCurrency Class Reference . . . . .	642
7.373	MakeMCDigitalEngine Class Template Reference . . . . .	643
7.374	MakeMCEuropeanEngine Class Template Reference . . . . .	644
7.375	MakeMCEuropeanHestonEngine Class Template Reference . . . . .	645
7.376	MakeSchedule Class Reference . . . . .	646
7.377	Matrix Class Reference . . . . .	647
7.378	MCAmericanBasketEngine Class Reference . . . . .	651
7.379	MCBarrierEngine Class Template Reference . . . . .	652
7.380	MCBasketEngine Class Template Reference . . . . .	654
7.381	McCliquetOption Class Reference . . . . .	656
7.382	MCDigitalEngine Class Template Reference . . . . .	657
7.383	MCDiscreteArithmeticAPEngine Class Template Reference . . . . .	658
7.384	McDiscreteArithmeticASO Class Reference . . . . .	660
7.385	MCDiscreteAveragingAsianEngine Class Template Reference . . . . .	661
7.386	MCDiscreteGeometricAPEngine Class Template Reference . . . . .	663
7.387	MCEuropeanEngine Class Template Reference . . . . .	664
7.388	MCEuropeanHestonEngine Class Template Reference . . . . .	665
7.389	McEverest Class Reference . . . . .	666
7.390	McHimalaya Class Reference . . . . .	667
7.391	McMaxBasket Class Reference . . . . .	668
7.392	McPagoda Class Reference . . . . .	669
7.393	McPerformanceOption Class Reference . . . . .	670
7.394	McPricer Class Template Reference . . . . .	671
7.395	McSimulation Class Template Reference . . . . .	672
7.396	MCVanillaEngine Class Template Reference . . . . .	674
7.397	MersenneTwisterUniformRng Class Reference . . . . .	675
7.398	Merton76Process Class Reference . . . . .	676
7.399	Mexico Class Reference . . . . .	678
7.400	MixedScheme Class Template Reference . . . . .	680
7.401	Money Class Reference . . . . .	682
7.402	MonotonicCubicSpline Class Reference . . . . .	684
7.403	MonteCarloModel Class Template Reference . . . . .	685
7.404	MoreGreeks Class Reference . . . . .	686

7.405	MoroInverseCumulativeNormal Class Reference . . . . .	687
7.406	MTLCurrency Class Reference . . . . .	688
7.407	MultiAssetOption Class Reference . . . . .	689
7.408	MultiAssetOption::arguments Class Reference . . . . .	691
7.409	MultiAssetOption::results Class Reference . . . . .	692
7.410	MultiCubicSpline Class Template Reference . . . . .	693
7.411	MultiPath Class Reference . . . . .	694
7.412	MultiPathGenerator Class Template Reference . . . . .	695
7.413	MultiVariate Struct Template Reference . . . . .	696
7.414	MXNCurrency Class Reference . . . . .	697
7.415	NaturalCubicSpline Class Reference . . . . .	698
7.416	NaturalMonotonicCubicSpline Class Reference . . . . .	699
7.417	NeumannBC Class Reference . . . . .	700
7.418	Newton Class Reference . . . . .	702
7.419	NewtonSafe Class Reference . . . . .	703
7.420	NewZealand Class Reference . . . . .	704
7.421	NLGCurrency Class Reference . . . . .	705
7.422	NoConstraint Class Reference . . . . .	706
7.423	NOKCurrency Class Reference . . . . .	707
7.424	NonLinearLeastSquare Class Reference . . . . .	708
7.425	NormalDistribution Class Reference . . . . .	709
7.426	Norway Class Reference . . . . .	710
7.427	NPRCurrency Class Reference . . . . .	711
7.428	Null Class Template Reference . . . . .	712
7.429	NullCalendar Class Reference . . . . .	713
7.430	NullCondition Class Template Reference . . . . .	714
7.431	NullParameter Class Reference . . . . .	715
7.432	NumericalMethod Class Reference . . . . .	716
7.433	NZDCurrency Class Reference . . . . .	718
7.434	NZDLibor Class Reference . . . . .	719
7.435	Observable Class Reference . . . . .	720
7.436	ObservableValue Class Template Reference . . . . .	721
7.437	Observer Class Reference . . . . .	722
7.438	OneAssetOption Class Reference . . . . .	724
7.439	OneAssetOption::arguments Class Reference . . . . .	727
7.440	OneAssetOption::results Class Reference . . . . .	728

7.441	OneAssetStrikedOption Class Reference . . . . .	729
7.442	OneDayCounter Class Reference . . . . .	731
7.443	OneFactorAffineModel Class Reference . . . . .	732
7.444	OneFactorModel Class Reference . . . . .	733
7.445	OneFactorModel::ShortRateDynamics Class Reference . . . . .	734
7.446	OneFactorModel::ShortRateTree Class Reference . . . . .	735
7.447	OperatorFactory Class Reference . . . . .	736
7.448	OptimizationMethod Class Reference . . . . .	737
7.449	Option Class Reference . . . . .	739
7.450	Option::arguments Class Reference . . . . .	740
7.451	OrnsteinUhlenbeckProcess Class Reference . . . . .	741
7.452	Parameter Class Reference . . . . .	743
7.453	ParameterImpl Class Reference . . . . .	744
7.454	ParCoupon Class Reference . . . . .	745
7.455	Path Class Reference . . . . .	747
7.456	PathGenerator Class Template Reference . . . . .	748
7.457	PathPricer Class Template Reference . . . . .	749
7.458	Payoff Class Reference . . . . .	750
7.459	PercentageStrikePayoff Class Reference . . . . .	751
7.460	Period Class Reference . . . . .	752
7.461	PiecewiseConstantParameter Class Reference . . . . .	753
7.462	PiecewiseYieldCurve Class Template Reference . . . . .	754
7.463	PKRCurrency Class Reference . . . . .	756
7.464	PlainVanillaPayoff Class Reference . . . . .	757
7.465	PLNCurrency Class Reference . . . . .	758
7.466	PoissonDistribution Class Reference . . . . .	759
7.467	Poland Class Reference . . . . .	760
7.468	PositiveConstraint Class Reference . . . . .	761
7.469	PriceCurve Class Reference . . . . .	762
7.470	PricingEngine Class Reference . . . . .	763
7.471	PrimeNumbers Class Reference . . . . .	764
7.472	Problem Class Reference . . . . .	765
7.473	PTECurrency Class Reference . . . . .	767
7.474	QuantoEngine Class Template Reference . . . . .	768
7.475	QuantoForwardVanillaOption Class Reference . . . . .	770
7.476	QuantoOptionArguments Class Template Reference . . . . .	772

7.477	QuantoOptionResults Class Template Reference . . . . .	773
7.478	QuantoTermStructure Class Reference . . . . .	774
7.479	QuantoVanillaOption Class Reference . . . . .	776
7.480	Quote Class Reference . . . . .	778
7.481	RandomizedLDS Class Template Reference . . . . .	779
7.482	RandomSequenceGenerator Class Template Reference . . . . .	781
7.483	RateHelper Class Reference . . . . .	782
7.484	Results Class Reference . . . . .	784
7.485	Ridder Class Reference . . . . .	785
7.486	ROLCurrency Class Reference . . . . .	786
7.487	Rounding Class Reference . . . . .	787
7.488	SalvagingAlgorithm Struct Reference . . . . .	789
7.489	Sample Struct Template Reference . . . . .	790
7.490	SampledCurve Class Reference . . . . .	791
7.491	SARCurrency Class Reference . . . . .	793
7.492	SaudiArabia Class Reference . . . . .	794
7.493	Schedule Class Reference . . . . .	795
7.494	Secant Class Reference . . . . .	796
7.495	SeedGenerator Class Reference . . . . .	797
7.496	SegmentIntegral Class Reference . . . . .	798
7.497	SEKCurrency Class Reference . . . . .	799
7.498	SequenceStatistics Class Template Reference . . . . .	800
7.499	Settings Class Reference . . . . .	802
7.500	SGDCurrency Class Reference . . . . .	804
7.501	Short Class Template Reference . . . . .	805
7.502	Short< ParCoupon > Class Template Reference . . . . .	806
7.503	ShortRateModel Class Reference . . . . .	807
7.504	ShoutCondition Class Reference . . . . .	809
7.505	SimpleCashFlow Class Reference . . . . .	810
7.506	SimpleDayCounter Class Reference . . . . .	812
7.507	SimpleQuote Class Reference . . . . .	813
7.508	Simplex Class Reference . . . . .	814
7.509	SimpsonIntegral Class Reference . . . . .	815
7.510	Singapore Class Reference . . . . .	816
7.511	SingleAssetOption Class Reference . . . . .	818
7.512	Singleton Class Template Reference . . . . .	820

7.513	SingleVariate Struct Template Reference . . . . .	821
7.514	SITCurrency Class Reference . . . . .	822
7.515	SKKCurrency Class Reference . . . . .	823
7.516	Slovakia Class Reference . . . . .	824
7.517	SobolRsg Class Reference . . . . .	826
7.518	Solver1D Class Template Reference . . . . .	828
7.519	SouthAfrica Class Reference . . . . .	830
7.520	SouthKorea Class Reference . . . . .	831
7.521	SquareRootProcess Class Reference . . . . .	833
7.522	StatsHolder Class Reference . . . . .	834
7.523	SteepestDescent Class Reference . . . . .	835
7.524	step_iterator Class Template Reference . . . . .	836
7.525	StepCondition Class Template Reference . . . . .	837
7.526	StepConditionSet Class Template Reference . . . . .	838
7.527	StochasticProcess Class Reference . . . . .	839
7.528	StochasticProcess1D Class Reference . . . . .	842
7.529	StochasticProcess1D::discretization Class Reference . . . . .	844
7.530	StochasticProcess::discretization Class Reference . . . . .	845
7.531	StochasticProcessArray Class Reference . . . . .	846
7.532	Stock Class Reference . . . . .	848
7.533	StrikedTypePayoff Class Reference . . . . .	849
7.534	StulzEngine Class Reference . . . . .	850
7.535	SuperSharePayoff Class Reference . . . . .	851
7.536	SVD Class Reference . . . . .	852
7.537	Swap Class Reference . . . . .	853
7.538	SwapRateHelper Class Reference . . . . .	855
7.539	Swaption Class Reference . . . . .	858
7.540	Swaption::arguments Class Reference . . . . .	860
7.541	Swaption::results Class Reference . . . . .	861
7.542	SwaptionHelper Class Reference . . . . .	862
7.543	SwaptionVolatilityMatrix Class Reference . . . . .	863
7.544	SwaptionVolatilityStructure Class Reference . . . . .	865
7.545	Sweden Class Reference . . . . .	867
7.546	Switzerland Class Reference . . . . .	868
7.547	SymmetricSchurDecomposition Class Reference . . . . .	869
7.548	TabulatedGaussLegendre Class Reference . . . . .	870

7.549	Taiwan Class Reference . . . . .	871
7.550	TARGET Class Reference . . . . .	873
7.551	TermStructure Class Reference . . . . .	874
7.552	TermStructureConsistentModel Class Reference . . . . .	876
7.553	TermStructureFittingParameter Class Reference . . . . .	877
7.554	THBCurrency Class Reference . . . . .	878
7.555	Thirty360 Class Reference . . . . .	879
7.556	Tian Class Reference . . . . .	880
7.557	Tibor Class Reference . . . . .	881
7.558	TimeBasket Class Reference . . . . .	882
7.559	TimeGrid Class Reference . . . . .	883
7.560	TqrEigenDecomposition Class Reference . . . . .	885
7.561	TransformedGrid Class Reference . . . . .	886
7.562	TrapezoidIntegral Class Reference . . . . .	887
7.563	Tree Class Template Reference . . . . .	889
7.564	TreeCapFloorEngine Class Reference . . . . .	890
7.565	TreeSwaptionEngine Class Reference . . . . .	891
7.566	TreeVanillaSwapEngine Class Reference . . . . .	892
7.567	TridiagonalOperator Class Reference . . . . .	893
7.568	TridiagonalOperator::TimeSetter Class Reference . . . . .	895
7.569	Trigeorgis Class Reference . . . . .	896
7.570	TrinomialTree Class Reference . . . . .	897
7.571	TRLCurrency Class Reference . . . . .	898
7.572	TRLibor Class Reference . . . . .	899
7.573	TRYCurrency Class Reference . . . . .	900
7.574	TsiveriotisFernandesLattice Class Template Reference . . . . .	901
7.575	TTDCurrency Class Reference . . . . .	903
7.576	Turkey Class Reference . . . . .	904
7.577	TWDCurrency Class Reference . . . . .	905
7.578	TwoFactorModel Class Reference . . . . .	906
7.579	TwoFactorModel::ShortRateDynamics Class Reference . . . . .	907
7.580	TwoFactorModel::ShortRateTree Class Reference . . . . .	908
7.581	TypePayoff Class Reference . . . . .	909
7.582	Ukraine Class Reference . . . . .	910
7.583	UnitedKingdom Class Reference . . . . .	912
7.584	UnitedStates Class Reference . . . . .	914



7.585	UpFrontIndexedCoupon Class Reference . . . . .	917
7.586	UpRounding Class Reference . . . . .	919
7.587	USDCurrency Class Reference . . . . .	920
7.588	USDLibor Class Reference . . . . .	921
7.589	Value Class Reference . . . . .	922
7.590	VanillaOption Class Reference . . . . .	923
7.591	VanillaOption::engine Class Reference . . . . .	924
7.592	VanillaSwap Class Reference . . . . .	925
7.593	VanillaSwap::arguments Class Reference . . . . .	927
7.594	VanillaSwap::results Class Reference . . . . .	928
7.595	Vasicek Class Reference . . . . .	929
7.596	Vasicek::Dynamics Class Reference . . . . .	931
7.597	VEBCurrency Class Reference . . . . .	932
7.598	Visitor Class Template Reference . . . . .	933
7.599	Xibor Class Reference . . . . .	934
7.600	YieldTermStructure Class Reference . . . . .	936
7.601	ZARCurrency Class Reference . . . . .	940
7.602	ZeroCondition Class Template Reference . . . . .	941
7.603	ZeroCouponBond Class Reference . . . . .	942
7.604	ZeroSpreadedTermStructure Class Reference . . . . .	943
7.605	ZeroYield Struct Reference . . . . .	945
7.606	ZeroYieldStructure Class Reference . . . . .	946
7.607	Zibor Class Reference . . . . .	948
<b>8</b>	<b>QuantLib File Documentation</b>	<b>949</b>
8.1	ql/argsandresults.hpp File Reference . . . . .	949
8.2	ql/calendar.hpp File Reference . . . . .	951
8.3	ql/Calendars/argentina.hpp File Reference . . . . .	953
8.4	ql/Calendars/beijing.hpp File Reference . . . . .	954
8.5	ql/Calendars/bombay.hpp File Reference . . . . .	955
8.6	ql/Calendars/bratislava.hpp File Reference . . . . .	956
8.7	ql/Calendars/brazil.hpp File Reference . . . . .	957
8.8	ql/Calendars/budapest.hpp File Reference . . . . .	958
8.9	ql/Calendars/copenhagen.hpp File Reference . . . . .	959
8.10	ql/Calendars/germany.hpp File Reference . . . . .	960
8.11	ql/Calendars/helsinki.hpp File Reference . . . . .	961
8.12	ql/Calendars/hongkong.hpp File Reference . . . . .	962

8.13	ql/Calendars/iceland.hpp File Reference . . . . .	963
8.14	ql/Calendars/indonesia.hpp File Reference . . . . .	964
8.15	ql/Calendars/istanbul.hpp File Reference . . . . .	965
8.16	ql/Calendars/italy.hpp File Reference . . . . .	966
8.17	ql/Calendars/johannesburg.hpp File Reference . . . . .	967
8.18	ql/Calendars/jointcalendar.hpp File Reference . . . . .	968
8.19	ql/Calendars/mexico.hpp File Reference . . . . .	969
8.20	ql/Calendars/nullcalendar.hpp File Reference . . . . .	970
8.21	ql/Calendars/oslo.hpp File Reference . . . . .	971
8.22	ql/Calendars/prague.hpp File Reference . . . . .	972
8.23	ql/Calendars/riyadh.hpp File Reference . . . . .	973
8.24	ql/Calendars/seoul.hpp File Reference . . . . .	974
8.25	ql/Calendars/singapore.hpp File Reference . . . . .	975
8.26	ql/Calendars/stockholm.hpp File Reference . . . . .	976
8.27	ql/Calendars/sydney.hpp File Reference . . . . .	977
8.28	ql/Calendars/taipei.hpp File Reference . . . . .	978
8.29	ql/Calendars/taiwan.hpp File Reference . . . . .	979
8.30	ql/Calendars/target.hpp File Reference . . . . .	980
8.31	ql/Calendars/tokyo.hpp File Reference . . . . .	981
8.32	ql/Calendars/toronto.hpp File Reference . . . . .	982
8.33	ql/Calendars/ukraine.hpp File Reference . . . . .	983
8.34	ql/Calendars/unitedkingdom.hpp File Reference . . . . .	984
8.35	ql/Calendars/unitedstates.hpp File Reference . . . . .	985
8.36	ql/Calendars/warsaw.hpp File Reference . . . . .	986
8.37	ql/Calendars/wellington.hpp File Reference . . . . .	987
8.38	ql/Calendars/zurich.hpp File Reference . . . . .	988
8.39	ql/capvolstructures.hpp File Reference . . . . .	989
8.40	ql/cashflow.hpp File Reference . . . . .	990
8.41	ql/CashFlows/analysis.hpp File Reference . . . . .	991
8.42	ql/CashFlows/basispointsensitivity.hpp File Reference . . . . .	992
8.43	ql/CashFlows/cashflowvectors.hpp File Reference . . . . .	993
8.44	ql/CashFlows/coupon.hpp File Reference . . . . .	994
8.45	ql/CashFlows/dividend.hpp File Reference . . . . .	995
8.46	ql/CashFlows/fixedratecoupon.hpp File Reference . . . . .	996
8.47	ql/CashFlows/floatingratecoupon.hpp File Reference . . . . .	997
8.48	ql/CashFlows/inarrearindexedcoupon.hpp File Reference . . . . .	998

8.49	ql/CashFlows/indexedcashflowvectors.hpp File Reference . . . . .	999
8.50	ql/CashFlows/indexedcoupon.hpp File Reference . . . . .	1000
8.51	ql/CashFlows/parcoupon.hpp File Reference . . . . .	1001
8.52	ql/CashFlows/shortfloatingcoupon.hpp File Reference . . . . .	1002
8.53	ql/CashFlows/shortindexedcoupon.hpp File Reference . . . . .	1003
8.54	ql/CashFlows/simplecashflow.hpp File Reference . . . . .	1004
8.55	ql/CashFlows/timebasket.hpp File Reference . . . . .	1005
8.56	ql/CashFlows/upfrontindexedcoupon.hpp File Reference . . . . .	1006
8.57	ql/Currencies/africa.hpp File Reference . . . . .	1007
8.58	ql/Currencies/america.hpp File Reference . . . . .	1008
8.59	ql/Currencies/asia.hpp File Reference . . . . .	1010
8.60	ql/Currencies/europe.hpp File Reference . . . . .	1012
8.61	ql/Currencies/exchangeratemanager.hpp File Reference . . . . .	1015
8.62	ql/Currencies/oceania.hpp File Reference . . . . .	1016
8.63	ql/currency.hpp File Reference . . . . .	1017
8.64	ql/date.hpp File Reference . . . . .	1018
8.65	ql/daycounter.hpp File Reference . . . . .	1021
8.66	ql/DayCounters/actual360.hpp File Reference . . . . .	1022
8.67	ql/DayCounters/actual365fixed.hpp File Reference . . . . .	1023
8.68	ql/DayCounters/actualactual.hpp File Reference . . . . .	1024
8.69	ql/DayCounters/one.hpp File Reference . . . . .	1025
8.70	ql/DayCounters/simpliedaycounter.hpp File Reference . . . . .	1026
8.71	ql/DayCounters/thirty360.hpp File Reference . . . . .	1027
8.72	ql/discretizedasset.hpp File Reference . . . . .	1028
8.73	ql/errors.hpp File Reference . . . . .	1029
8.74	ql/event.hpp File Reference . . . . .	1032
8.75	ql/exchangerate.hpp File Reference . . . . .	1033
8.76	ql/exercise.hpp File Reference . . . . .	1034
8.77	ql/FiniteDifferences/americancondition.hpp File Reference . . . . .	1035
8.78	ql/FiniteDifferences/boundarycondition.hpp File Reference . . . . .	1036
8.79	ql/FiniteDifferences/bsmoperator.hpp File Reference . . . . .	1037
8.80	ql/FiniteDifferences/bsmtermoperator.hpp File Reference . . . . .	1038
8.81	ql/FiniteDifferences/cranknicolson.hpp File Reference . . . . .	1039
8.82	ql/FiniteDifferences/dminus.hpp File Reference . . . . .	1040
8.83	ql/FiniteDifferences/dplus.hpp File Reference . . . . .	1041
8.84	ql/FiniteDifferences/dplusdminus.hpp File Reference . . . . .	1042

8.85	ql/FiniteDifferences/dzero.hpp File Reference . . . . .	1043
8.86	ql/FiniteDifferences/expliciteuler.hpp File Reference . . . . .	1044
8.87	ql/FiniteDifferences/fdtypedefs.hpp File Reference . . . . .	1045
8.88	ql/FiniteDifferences/finitedifferencemodel.hpp File Reference . . . . .	1046
8.89	ql/FiniteDifferences/impliciteuler.hpp File Reference . . . . .	1047
8.90	ql/FiniteDifferences/mixedscheme.hpp File Reference . . . . .	1048
8.91	ql/FiniteDifferences/onefactoroperator.hpp File Reference . . . . .	1049
8.92	ql/FiniteDifferences/operatorfactory.hpp File Reference . . . . .	1050
8.93	ql/FiniteDifferences/operatortraits.hpp File Reference . . . . .	1051
8.94	ql/FiniteDifferences/parallelevolver.hpp File Reference . . . . .	1052
8.95	ql/FiniteDifferences/pde.hpp File Reference . . . . .	1053
8.96	ql/FiniteDifferences/pdebsm.hpp File Reference . . . . .	1054
8.97	ql/FiniteDifferences/pdeshortrate.hpp File Reference . . . . .	1055
8.98	ql/FiniteDifferences/shoutcondition.hpp File Reference . . . . .	1056
8.99	ql/FiniteDifferences/stepcondition.hpp File Reference . . . . .	1057
8.100	ql/FiniteDifferences/tridiagonaloperator.hpp File Reference . . . . .	1058
8.101	ql/FiniteDifferences/valueatcenter.hpp File Reference . . . . .	1059
8.102	ql/FiniteDifferences/zerocondition.hpp File Reference . . . . .	1060
8.103	ql/grid.hpp File Reference . . . . .	1061
8.104	ql/handle.hpp File Reference . . . . .	1062
8.105	ql/history.hpp File Reference . . . . .	1063
8.106	ql/index.hpp File Reference . . . . .	1064
8.107	ql/Indexes/audlibor.hpp File Reference . . . . .	1065
8.108	ql/Indexes/cadlibor.hpp File Reference . . . . .	1066
8.109	ql/Indexes/cdor.hpp File Reference . . . . .	1067
8.110	ql/Indexes/chflibor.hpp File Reference . . . . .	1068
8.111	ql/Indexes/dkklibor.hpp File Reference . . . . .	1069
8.112	ql/Indexes/euribor.hpp File Reference . . . . .	1070
8.113	ql/Indexes/eurlibor.hpp File Reference . . . . .	1071
8.114	ql/Indexes/gbplibor.hpp File Reference . . . . .	1072
8.115	ql/Indexes/indexmanager.hpp File Reference . . . . .	1073
8.116	ql/Indexes/jibar.hpp File Reference . . . . .	1074
8.117	ql/Indexes/jpylibor.hpp File Reference . . . . .	1075
8.118	ql/Indexes/libor.hpp File Reference . . . . .	1076
8.119	ql/Indexes/nzdlibor.hpp File Reference . . . . .	1077
8.120	ql/Indexes/tibor.hpp File Reference . . . . .	1078

8.121	ql/Indexes/trlibor.hpp File Reference . . . . .	1079
8.122	ql/Indexes/usdlibor.hpp File Reference . . . . .	1080
8.123	ql/Indexes/xibor.hpp File Reference . . . . .	1081
8.124	ql/Indexes/zibor.hpp File Reference . . . . .	1082
8.125	ql/instrument.hpp File Reference . . . . .	1083
8.126	ql/Instruments/asianooption.hpp File Reference . . . . .	1084
8.127	ql/Instruments/barrierooption.hpp File Reference . . . . .	1085
8.128	ql/Instruments/basketoption.hpp File Reference . . . . .	1086
8.129	ql/Instruments/bond.hpp File Reference . . . . .	1087
8.130	ql/Instruments/callabilityschedule.hpp File Reference . . . . .	1088
8.131	ql/Instruments/capfloor.hpp File Reference . . . . .	1089
8.132	ql/Instruments/cliquestoption.hpp File Reference . . . . .	1090
8.133	ql/Instruments/convertiblebond.hpp File Reference . . . . .	1091
8.134	ql/Instruments/dividendschedule.hpp File Reference . . . . .	1093
8.135	ql/Instruments/dividendvanillaoption.hpp File Reference . . . . .	1094
8.136	ql/Instruments/europeanoption.hpp File Reference . . . . .	1095
8.137	ql/Instruments/fixedcouponbond.hpp File Reference . . . . .	1096
8.138	ql/Instruments/floatingratebond.hpp File Reference . . . . .	1097
8.139	ql/Instruments/forwardvanillaoption.hpp File Reference . . . . .	1098
8.140	ql/Instruments/multiassetoption.hpp File Reference . . . . .	1099
8.141	ql/Instruments/oneassetoption.hpp File Reference . . . . .	1100
8.142	ql/Instruments/oneassetstrikedoption.hpp File Reference . . . . .	1101
8.143	ql/Instruments/payoffs.hpp File Reference . . . . .	1102
8.144	ql/Instruments/quantoforwardvanillaoption.hpp File Reference . . . . .	1103
8.145	ql/Instruments/quantovanillaoption.hpp File Reference . . . . .	1104
8.146	ql/Instruments/simpleswap.hpp File Reference . . . . .	1105
8.147	ql/Instruments/stock.hpp File Reference . . . . .	1106
8.148	ql/Instruments/swap.hpp File Reference . . . . .	1107
8.149	ql/Instruments/swaption.hpp File Reference . . . . .	1108
8.150	ql/Instruments/vanillaoption.hpp File Reference . . . . .	1109
8.151	ql/Instruments/zerocouponbond.hpp File Reference . . . . .	1110
8.152	ql/interestrates.hpp File Reference . . . . .	1111
8.153	ql/Lattices/binomialtree.hpp File Reference . . . . .	1112
8.154	ql/Lattices/bsmllattice.hpp File Reference . . . . .	1114
8.155	ql/Lattices/lattice.hpp File Reference . . . . .	1115
8.156	ql/Lattices/lattice1d.hpp File Reference . . . . .	1116

8.157	ql/Lattices/lattice2d.hpp File Reference . . . . .	1117
8.158	ql/Lattices/tflattice.hpp File Reference . . . . .	1118
8.159	ql/Lattices/tree.hpp File Reference . . . . .	1119
8.160	ql/Lattices/trinomialtree.hpp File Reference . . . . .	1120
8.161	ql/Math/array.hpp File Reference . . . . .	1121
8.162	ql/Math/backwardflatinterpolation.hpp File Reference . . . . .	1122
8.163	ql/Math/beta.hpp File Reference . . . . .	1123
8.164	ql/Math/bicubicsplineinterpolation.hpp File Reference . . . . .	1124
8.165	ql/Math/bilinearinterpolation.hpp File Reference . . . . .	1125
8.166	ql/Math/binomialdistribution.hpp File Reference . . . . .	1126
8.167	ql/Math/bivariatenormaldistribution.hpp File Reference . . . . .	1127
8.168	ql/Math/chisquaredistribution.hpp File Reference . . . . .	1128
8.169	ql/Math/choleskydecomposition.hpp File Reference . . . . .	1129
8.170	ql/Math/comparison.hpp File Reference . . . . .	1130
8.171	ql/Math/convergencestatistics.hpp File Reference . . . . .	1131
8.172	ql/Math/cubicspline.hpp File Reference . . . . .	1132
8.173	ql/Math/discrepancystatistics.hpp File Reference . . . . .	1133
8.174	ql/Math/errorfunction.hpp File Reference . . . . .	1134
8.175	ql/Math/extrapolation.hpp File Reference . . . . .	1135
8.176	ql/Math/factorial.hpp File Reference . . . . .	1136
8.177	ql/Math/forwardflatinterpolation.hpp File Reference . . . . .	1137
8.178	ql/Math/functional.hpp File Reference . . . . .	1138
8.179	ql/Math/gammadistribution.hpp File Reference . . . . .	1139
8.180	ql/Math/gaussianorthogonalpolynomial.hpp File Reference . . . . .	1140
8.181	ql/Math/gaussianquadratures.hpp File Reference . . . . .	1141
8.182	ql/Math/gaussianstatistics.hpp File Reference . . . . .	1143
8.183	ql/Math/generalstatistics.hpp File Reference . . . . .	1144
8.184	ql/Math/incompletegamma.hpp File Reference . . . . .	1145
8.185	ql/Math/incrementalstatistics.hpp File Reference . . . . .	1146
8.186	ql/Math/interpolation.hpp File Reference . . . . .	1147
8.187	ql/Math/interpolation2D.hpp File Reference . . . . .	1148
8.188	ql/Math/kronrodintegral.hpp File Reference . . . . .	1149
8.189	ql/Math/lexicographicalview.hpp File Reference . . . . .	1150
8.190	ql/Math/linearinterpolation.hpp File Reference . . . . .	1151
8.191	ql/Math/loglinearinterpolation.hpp File Reference . . . . .	1152
8.192	ql/Math/matrix.hpp File Reference . . . . .	1153

8.193	ql/Math/multicubicspline.hpp File Reference . . . . .	1154
8.194	ql/Math/normaldistribution.hpp File Reference . . . . .	1156
8.195	ql/Math/poissondistribution.hpp File Reference . . . . .	1157
8.196	ql/Math/primenumbers.hpp File Reference . . . . .	1158
8.197	ql/Math/pseudosqrt.hpp File Reference . . . . .	1159
8.198	ql/Math/riskstatistics.hpp File Reference . . . . .	1160
8.199	ql/Math/rounding.hpp File Reference . . . . .	1161
8.200	ql/Math/sampledcurve.hpp File Reference . . . . .	1162
8.201	ql/Math/segmentintegral.hpp File Reference . . . . .	1163
8.202	ql/Math/sequencestatistics.hpp File Reference . . . . .	1164
8.203	ql/Math/simpsonintegral.hpp File Reference . . . . .	1166
8.204	ql/Math/statistics.hpp File Reference . . . . .	1167
8.205	ql/Math/svd.hpp File Reference . . . . .	1168
8.206	ql/Math/symmetriceigenvalues.hpp File Reference . . . . .	1169
8.207	ql/Math/symmetricschurdecomposition.hpp File Reference . . . . .	1170
8.208	ql/Math/tqreigendecomposition.hpp File Reference . . . . .	1171
8.209	ql/Math/transformedgrid.hpp File Reference . . . . .	1172
8.210	ql/Math/trapezoidintegral.hpp File Reference . . . . .	1173
8.211	ql/money.hpp File Reference . . . . .	1174
8.212	ql/MonteCarlo/brownianbridge.hpp File Reference . . . . .	1175
8.213	ql/MonteCarlo/getcovariance.hpp File Reference . . . . .	1176
8.214	ql/MonteCarlo/mctraits.hpp File Reference . . . . .	1177
8.215	ql/MonteCarlo/mctypedefs.hpp File Reference . . . . .	1178
8.216	ql/MonteCarlo/montecarlomodel.hpp File Reference . . . . .	1179
8.217	ql/MonteCarlo/multipath.hpp File Reference . . . . .	1180
8.218	ql/MonteCarlo/multipathgenerator.hpp File Reference . . . . .	1181
8.219	ql/MonteCarlo/path.hpp File Reference . . . . .	1182
8.220	ql/MonteCarlo/pathgenerator.hpp File Reference . . . . .	1183
8.221	ql/MonteCarlo/pathpricer.hpp File Reference . . . . .	1184
8.222	ql/MonteCarlo/sample.hpp File Reference . . . . .	1185
8.223	ql/numericalmethod.hpp File Reference . . . . .	1186
8.224	ql/Optimization/armijo.hpp File Reference . . . . .	1187
8.225	ql/Optimization/conjugategradient.hpp File Reference . . . . .	1188
8.226	ql/Optimization/constraint.hpp File Reference . . . . .	1189
8.227	ql/Optimization/costfunction.hpp File Reference . . . . .	1190
8.228	ql/Optimization/criteria.hpp File Reference . . . . .	1191

8.229	ql/Optimization/leastsquare.hpp File Reference . . . . .	1192
8.230	ql/Optimization/levenbergmarquardt.hpp File Reference . . . . .	1193
8.231	ql/Optimization/linesearch.hpp File Reference . . . . .	1194
8.232	ql/Optimization/lmdif.hpp File Reference . . . . .	1195
8.233	ql/Optimization/method.hpp File Reference . . . . .	1196
8.234	ql/Optimization/problem.hpp File Reference . . . . .	1197
8.235	ql/Optimization/simplex.hpp File Reference . . . . .	1198
8.236	ql/Optimization/steepestdescent.hpp File Reference . . . . .	1199
8.237	ql/option.hpp File Reference . . . . .	1200
8.238	ql/Patterns/bridge.hpp File Reference . . . . .	1201
8.239	ql/Patterns/composite.hpp File Reference . . . . .	1202
8.240	ql/Patterns/curiouslyrecurring.hpp File Reference . . . . .	1203
8.241	ql/Patterns/lazyobject.hpp File Reference . . . . .	1204
8.242	ql/Patterns/observable.hpp File Reference . . . . .	1205
8.243	ql/Patterns/singleton.hpp File Reference . . . . .	1206
8.244	ql/Patterns/visitor.hpp File Reference . . . . .	1207
8.245	ql/payoff.hpp File Reference . . . . .	1208
8.246	ql/Pricers/discretegeometriccaso.hpp File Reference . . . . .	1209
8.247	ql/Pricers/mccliquestoption.hpp File Reference . . . . .	1210
8.248	ql/Pricers/mcdiscretearithmeticaso.hpp File Reference . . . . .	1211
8.249	ql/Pricers/mceverest.hpp File Reference . . . . .	1212
8.250	ql/Pricers/mchimalaya.hpp File Reference . . . . .	1213
8.251	ql/Pricers/mcmaxbasket.hpp File Reference . . . . .	1214
8.252	ql/Pricers/mcpagoda.hpp File Reference . . . . .	1215
8.253	ql/Pricers/mcperformanceoption.hpp File Reference . . . . .	1216
8.254	ql/Pricers/mcpricer.hpp File Reference . . . . .	1217
8.255	ql/Pricers/singleassetoption.hpp File Reference . . . . .	1218
8.256	ql/pricingengine.hpp File Reference . . . . .	1219
8.257	ql/PricingEngines/americanpayoffatexpiry.hpp File Reference . . . . .	1220
8.258	ql/PricingEngines/americanpayoffathit.hpp File Reference . . . . .	1221
8.259	ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp File Reference . . . .	1222
8.260	ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp File Reference . . .	1223
8.261	ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp File Reference . . . . .	1224
8.262	ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp File Reference . . . . .	1225
8.263	ql/PricingEngines/Asian/mcdiscreteasianengine.hpp File Reference . . . . .	1226
8.264	ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference . . . . .	1227



8.265	ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference . . . . .	1228
8.266	ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference . . . . .	1229
8.267	ql/PricingEngines/Basket/mcbasketengine.hpp File Reference . . . . .	1230
8.268	ql/PricingEngines/Basket/stulzengine.hpp File Reference . . . . .	1231
8.269	ql/PricingEngines/blackformula.hpp File Reference . . . . .	1232
8.270	ql/PricingEngines/blackmodel.hpp File Reference . . . . .	1233
8.271	ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference . . . . .	1234
8.272	ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference . . . . .	1235
8.273	ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference . . . . .	1236
8.274	ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference . . . . .	1237
8.275	ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference . . . . .	1238
8.276	ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference . . . . .	1239
8.277	ql/PricingEngines/Forward/forwardengine.hpp File Reference . . . . .	1240
8.278	ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference . . . . .	1241
8.279	ql/PricingEngines/genericmodelengine.hpp File Reference . . . . .	1242
8.280	ql/PricingEngines/greeks.hpp File Reference . . . . .	1243
8.281	ql/PricingEngines/Hybrid/binomialconvertibleengine.hpp File Reference . . . . .	1244
8.282	ql/PricingEngines/Hybrid/discretizedconvertible.hpp File Reference . . . . .	1245
8.283	ql/PricingEngines/latticeshortratemodelengine.hpp File Reference . . . . .	1246
8.284	ql/PricingEngines/mcsimulation.hpp File Reference . . . . .	1247
8.285	ql/PricingEngines/Quanto/quantoengine.hpp File Reference . . . . .	1248
8.286	ql/PricingEngines/Swaption/blackswaptionengine.hpp File Reference . . . . .	1249
8.287	ql/PricingEngines/Swaption/discretizedswaption.hpp File Reference . . . . .	1250
8.288	ql/PricingEngines/Swaption/g2swaptionengine.hpp File Reference . . . . .	1251
8.289	ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp File Reference . . . . .	1252
8.290	ql/PricingEngines/Swaption/lfmwaptionengine.hpp File Reference . . . . .	1253
8.291	ql/PricingEngines/Swaption/treeswaptionengine.hpp File Reference . . . . .	1254
8.292	ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference . . . . .	1255
8.293	ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp File Reference . . . . .	1256
8.294	ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference . . . . .	1257
8.295	ql/PricingEngines/Vanilla/analytichestonengine.hpp File Reference . . . . .	1258
8.296	ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference . . . . .	1259
8.297	ql/PricingEngines/Vanilla/batesengine.hpp File Reference . . . . .	1260
8.298	ql/PricingEngines/Vanilla/binomialengine.hpp File Reference . . . . .	1261
8.299	ql/PricingEngines/Vanilla/bjerkhundstenglandengine.hpp File Reference . . . . .	1262
8.300	ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference . . . . .	1263

8.301	ql/PricingEngines/Vanilla/fdamericanengine.hpp File Reference . . . . .	1264
8.302	ql/PricingEngines/Vanilla/fdbermudanengine.hpp File Reference . . . . .	1265
8.303	ql/PricingEngines/Vanilla/fdconditions.hpp File Reference . . . . .	1266
8.304	ql/PricingEngines/Vanilla/fddividendamericanengine.hpp File Reference . . . . .	1267
8.305	ql/PricingEngines/Vanilla/fddividendengine.hpp File Reference . . . . .	1268
8.306	ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp File Reference . . . . .	1269
8.307	ql/PricingEngines/Vanilla/fddividendshoutengine.hpp File Reference . . . . .	1270
8.308	ql/PricingEngines/Vanilla/fdeuropeanengine.hpp File Reference . . . . .	1271
8.309	ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp File Reference . . . . .	1272
8.310	ql/PricingEngines/Vanilla/fdshoutengine.hpp File Reference . . . . .	1273
8.311	ql/PricingEngines/Vanilla/fdstepconditionengine.hpp File Reference . . . . .	1274
8.312	ql/PricingEngines/Vanilla/fdvanillaengine.hpp File Reference . . . . .	1275
8.313	ql/PricingEngines/Vanilla/integralengine.hpp File Reference . . . . .	1276
8.314	ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference . . . . .	1277
8.315	ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference . . . . .	1278
8.316	ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference . . . . .	1279
8.317	ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference . . . . .	1280
8.318	ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp File Reference . . . . .	1281
8.319	ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference . . . . .	1282
8.320	ql/Processes/blackscholesprocess.hpp File Reference . . . . .	1283
8.321	ql/Processes/eulerdiscretization.hpp File Reference . . . . .	1284
8.322	ql/Processes/geometricbrownianprocess.hpp File Reference . . . . .	1285
8.323	ql/Processes/hestonprocess.hpp File Reference . . . . .	1286
8.324	ql/Processes/lfmcovarparam.hpp File Reference . . . . .	1287
8.325	ql/Processes/lfmhullwhiteparam.hpp File Reference . . . . .	1288
8.326	ql/Processes/lfmprocess.hpp File Reference . . . . .	1289
8.327	ql/Processes/merton76process.hpp File Reference . . . . .	1290
8.328	ql/Processes/ornsteinuhlenbeckprocess.hpp File Reference . . . . .	1291
8.329	ql/Processes/squarerootprocess.hpp File Reference . . . . .	1292
8.330	ql/Processes/stochasticprocessarray.hpp File Reference . . . . .	1293
8.331	ql/qldefines.hpp File Reference . . . . .	1294
8.332	ql/quote.hpp File Reference . . . . .	1296
8.333	ql/RandomNumbers/boxmullergaussianrng.hpp File Reference . . . . .	1297
8.334	ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference . . . . .	1298
8.335	ql/RandomNumbers/faurersg.hpp File Reference . . . . .	1299
8.336	ql/RandomNumbers/haltonrng.hpp File Reference . . . . .	1300

8.337	ql/RandomNumbers/inversecumulativerng.hpp File Reference . . . . .	1301
8.338	ql/RandomNumbers/inversecumulativersg.hpp File Reference . . . . .	1302
8.339	ql/RandomNumbers/knuthuniformrng.hpp File Reference . . . . .	1303
8.340	ql/RandomNumbers/lecuyeruniformrng.hpp File Reference . . . . .	1304
8.341	ql/RandomNumbers/mt19937uniformrng.hpp File Reference . . . . .	1305
8.342	ql/RandomNumbers/randomizedlds.hpp File Reference . . . . .	1306
8.343	ql/RandomNumbers/randomsequencegenerator.hpp File Reference . . . . .	1307
8.344	ql/RandomNumbers/rngtraits.hpp File Reference . . . . .	1308
8.345	ql/RandomNumbers/seedgenerator.hpp File Reference . . . . .	1310
8.346	ql/RandomNumbers/sobolrng.hpp File Reference . . . . .	1311
8.347	ql/schedule.hpp File Reference . . . . .	1312
8.348	ql/settings.hpp File Reference . . . . .	1313
8.349	ql/ShortRateModels/calibrationhelper.hpp File Reference . . . . .	1314
8.350	ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference . . . . .	1315
8.351	ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp File Reference . . . . .	1316
8.352	ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference . . . . .	1317
8.353	ql/ShortRateModels/LiborMarketModels/lfmcovarproxy.hpp File Reference . . . . .	1318
8.354	ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp File Reference . . . . .	1319
8.355	ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp File Reference . . . . .	1320
8.356	ql/ShortRateModels/LiborMarketModels/lmexpcorrmodel.hpp File Reference . . . . .	1321
8.357	ql/ShortRateModels/LiborMarketModels/lmfixedvolmodel.hpp File Reference . . . . .	1322
8.358	ql/ShortRateModels/LiborMarketModels/lmlnexpvolmodel.hpp File Reference . . . . .	1323
8.359	ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp File Reference . . . . .	1324
8.360	ql/ShortRateModels/model.hpp File Reference . . . . .	1325
8.361	ql/ShortRateModels/onefactormodel.hpp File Reference . . . . .	1326
8.362	ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference . . . . .	1327
8.363	ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference . . . . .	1328
8.364	ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference . . . . .	1329
8.365	ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference . . . . .	1330
8.366	ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference . . . . .	1331
8.367	ql/ShortRateModels/parameter.hpp File Reference . . . . .	1332
8.368	ql/ShortRateModels/twofactormodel.hpp File Reference . . . . .	1333
8.369	ql/ShortRateModels/TwoFactorModels/batesmodel.hpp File Reference . . . . .	1334
8.370	ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference . . . . .	1335
8.371	ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp File Reference . . . . .	1336
8.372	ql/solver1d.hpp File Reference . . . . .	1337

8.373	ql/Solvers1D/bisection.hpp File Reference . . . . .	1338
8.374	ql/Solvers1D/brent.hpp File Reference . . . . .	1339
8.375	ql/Solvers1D/falseposition.hpp File Reference . . . . .	1340
8.376	ql/Solvers1D/newton.hpp File Reference . . . . .	1341
8.377	ql/Solvers1D/newtonsafe.hpp File Reference . . . . .	1342
8.378	ql/Solvers1D/ridder.hpp File Reference . . . . .	1343
8.379	ql/Solvers1D/secant.hpp File Reference . . . . .	1344
8.380	ql/stochasticprocess.hpp File Reference . . . . .	1345
8.381	ql/swaptionvolstructure.hpp File Reference . . . . .	1346
8.382	ql/termstructure.hpp File Reference . . . . .	1347
8.383	ql/TermStructures/affinetermstructure.hpp File Reference . . . . .	1348
8.384	ql/TermStructures/bondhelpers.hpp File Reference . . . . .	1349
8.385	ql/TermStructures/bootstraptraits.hpp File Reference . . . . .	1350
8.386	ql/TermStructures/compoundforward.hpp File Reference . . . . .	1351
8.387	ql/TermStructures/discountcurve.hpp File Reference . . . . .	1352
8.388	ql/TermStructures/drifttermstructure.hpp File Reference . . . . .	1353
8.389	ql/TermStructures/extendeddiscountcurve.hpp File Reference . . . . .	1354
8.390	ql/TermStructures/flatforward.hpp File Reference . . . . .	1355
8.391	ql/TermStructures/forwardcurve.hpp File Reference . . . . .	1356
8.392	ql/TermStructures/forwardspreadedtermstructure.hpp File Reference . . . . .	1357
8.393	ql/TermStructures/forwardstructure.hpp File Reference . . . . .	1358
8.394	ql/TermStructures/impliedtermstructure.hpp File Reference . . . . .	1359
8.395	ql/TermStructures/piecewiseflatforward.hpp File Reference . . . . .	1360
8.396	ql/TermStructures/piecewiseyieldcurve.hpp File Reference . . . . .	1361
8.397	ql/TermStructures/quantotermstructure.hpp File Reference . . . . .	1362
8.398	ql/TermStructures/ratehelpers.hpp File Reference . . . . .	1363
8.399	ql/TermStructures/zerocurve.hpp File Reference . . . . .	1364
8.400	ql/TermStructures/zerospreadedtermstructure.hpp File Reference . . . . .	1365
8.401	ql/TermStructures/zeroyieldstructure.hpp File Reference . . . . .	1366
8.402	ql/timegrid.hpp File Reference . . . . .	1367
8.403	ql/types.hpp File Reference . . . . .	1368
8.404	ql/Utilities/dataformatters.hpp File Reference . . . . .	1370
8.405	ql/Utilities/dataparsers.hpp File Reference . . . . .	1371
8.406	ql/Utilities/disposable.hpp File Reference . . . . .	1372
8.407	ql/Utilities/null.hpp File Reference . . . . .	1373
8.408	ql/Utilities/observablevalue.hpp File Reference . . . . .	1374

8.409	ql/Utilities/steppingiterator.hpp File Reference . . . . .	1375
8.410	ql/Utilities/strings.hpp File Reference . . . . .	1376
8.411	ql/Utilities/tracing.hpp File Reference . . . . .	1377
8.412	ql/Volatilities/blackconstantvol.hpp File Reference . . . . .	1379
8.413	ql/Volatilities/blackvariancecurve.hpp File Reference . . . . .	1380
8.414	ql/Volatilities/blackvariancesurface.hpp File Reference . . . . .	1381
8.415	ql/Volatilities/capflatvolvector.hpp File Reference . . . . .	1382
8.416	ql/Volatilities/capletconstantvol.hpp File Reference . . . . .	1383
8.417	ql/Volatilities/capletvariancecurve.hpp File Reference . . . . .	1384
8.418	ql/Volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp File Reference . . . . .	1385
8.419	ql/Volatilities/localconstantvol.hpp File Reference . . . . .	1386
8.420	ql/Volatilities/localvolcurve.hpp File Reference . . . . .	1387
8.421	ql/Volatilities/localvolsurface.hpp File Reference . . . . .	1388
8.422	ql/Volatilities/swaptionvolmatrix.hpp File Reference . . . . .	1389
8.423	ql/voltermstructure.hpp File Reference . . . . .	1390
8.424	ql/yieldtermstructure.hpp File Reference . . . . .	1391
<b>9</b>	<b>QuantLib Example Documentation</b>	<b>1393</b>
9.1	BermudanSwaption.cpp . . . . .	1393
9.2	ConvertibleBonds.cpp . . . . .	1399
9.3	DiscreteHedging.cpp . . . . .	1404
9.4	EquityOption.cpp . . . . .	1410
9.5	history_iterators.cpp . . . . .	1415
9.6	swapvaluation.cpp . . . . .	1416
9.7	tracing_example.cpp . . . . .	1428
<b>10</b>	<b>Caveats</b>	<b>1431</b>
<b>11</b>	<b>Test Suite</b>	<b>1439</b>
<b>12</b>	<b>Todo List</b>	<b>1451</b>
<b>13</b>	<b>Known Bugs</b>	<b>1457</b>
<b>14</b>	<b>Deprecated List</b>	<b>1459</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.

To do:

- Fix finite difference in presence of dividends
- All pricers should be moved to the pricing engine framework.

### 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 prevent my auto-responder from posting to the list?

### QuantLib cross-platform support/extensions

- Does QuantLib support .NET?
- What's the difference between QuantLibXL and QuantLibAddin?

### 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.

At the time of this writing, it is somewhat difficult to modify the include path of the latest Visual C++ Express Beta. Instructions to perform such operation are available at <http://forums.microsoft.com/msdn/ShowPost.aspx?PostID=2995>.

**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 troubleshooting page; see [<http://quantlib.org/quantlibaddin/troubleshooting.html>](http://quantlib.org/quantlibaddin/troubleshooting.html).

### 1.7.8 QuantLib mailing lists

**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>](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.

#### What's the difference between QuantLibXL and QuantLibAddin?

Feature	QuantLibXL	QuantLibAddin
<b>Design</b>	Procedural and stateless. Each function returns a value, no state persists within the library	Object oriented and stateful. Calling a function instantiates an object in the repository - objects may be interrogated, updated, or passed as inputs to constructors/member functions of other objects.
<b>Lifecycle</b>	Stable, discontinued. Last updated for QuantLib 0.3.8	Prototype which at the time of this writing is under active development
<b>Platforms</b>	Microsoft Excel	Microsoft Excel, OpenOffice.org Calc (Windows and Linux), Guile, C/C++

## 1.8 Version history

### Release 0.3.12 - March 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-spread and forward-spread 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 Luttmmer. 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, Monte Paschi Asset Management sgr, administrator
- Luigi Ballabio, StatPro Italia srl, administrator
- Mario Aleppo, StatPro Italia srl
- Nicolas Di Césaré
- Dirk Eddelbuettel
- Neil Firth, Mathematical Institute, University of Oxford
- André Louw, Decillion Pty
- Marco Marchioro, StatPro Italia srl
- Sadruddin Rejeb
- Niels Elken Sønderby
- Enrico Sirola, StatPro Italia srl
- Ligu Song
- Joseph Wang

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

### 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, Antoine Cellier, Aurelien Chanudet, Jon Davidson, Daniele De Francesco, Piter Dias, Matteo Gallivanoni, Roman Gitlin, Tomoya Kawanishi, Gary Kennedy, Enrico Michelotti, Gilbert Pfeffer, Walter Penschke, Gianni Piolanti, Peter Schmitteckert, David Schwartz, Maxim Sokolov, Klaus Spanderen, Marco Tarengi, Charles Whitmore, Bernd Johannes Wuebben, and Jeff Yu.

This product 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 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 Sønderby  
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 Boafo  
Copyright (C) 2005 Piter Dias  
Copyright (C) 2005 Gary Kennedy  
Copyright (C) 2005 Joseph Wang  
Copyright (C) 2005 Charles Whitmore

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 . . . . .	87
Currencies and FX rates . . . . .	89
Date and time calculations . . . . .	93
Calendars . . . . .	96
Day counters . . . . .	99
Pricing engines . . . . .	100
Asian option engines . . . . .	101
Barrier option engines . . . . .	102
Basket option engines . . . . .	103
Cap/floor engines . . . . .	104
Cliquet option engines . . . . .	105
Forward option engines . . . . .	106
Quanto option engines . . . . .	107
Swaption engines . . . . .	108
Vanilla option engines . . . . .	109
Finite-differences framework . . . . .	112
Short-rate modelling framework . . . . .	119
Financial instruments . . . . .	122
Lattice methods . . . . .	125
Math tools . . . . .	128
Monte Carlo framework . . . . .	130
Design patterns . . . . .	136
Term structures . . . . .	137
Utilities . . . . .	139
QuantLib macros . . . . .	141
Generic macros . . . . .	142
Numeric limits . . . . .	143
Template capabilities . . . . .	144
Iterator support . . . . .	145
Debugging macros . . . . .	147
Output manipulators . . . . .	146



## Chapter 3

# QuantLib Hierarchical Index

### 3.1 QuantLib Class Hierarchy

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

AcyclicVisitor . . . . .	154
BPSBasketCalculator . . . . .	250
AmericanCondition . . . . .	159
AmericanPayoffAtExpiry . . . . .	161
AmericanPayoffAtHit . . . . .	162
Arguments . . . . .	175
CapFloor::arguments . . . . .	274
Option::arguments . . . . .	740
MultiAssetOption::arguments . . . . .	691
BasketOption::arguments . . . . .	197
OneAssetOption::arguments . . . . .	727
Swaption::arguments . . . . .	860
VanillaSwap::arguments . . . . .	927
Swaption::arguments . . . . .	860
Array . . . . .	177
Average . . . . .	186
BackwardFlat . . . . .	187
Barrier . . . . .	190
BarrierOption::arguments . . . . .	193
Bicubic . . . . .	206
Bilinear . . . . .	208
BinomialDistribution . . . . .	211
BivariateCumulativeNormalDistributionDr78 . . . . .	215
BivariateCumulativeNormalDistributionWe04DP . . . . .	216
BlackFormula . . . . .	221
BlackKarasinski::Dynamics . . . . .	224
BoundaryCondition . . . . .	246
BoundaryCondition< TridiagonalOperator > . . . . .	246
DirichletBC . . . . .	362
NeumannBC . . . . .	700
BoxMullerGaussianRng . . . . .	249
Bridge . . . . .	253

Bridge< Calendar, CalendarImpl > . . . . .	253
Calendar . . . . .	260
Argentina . . . . .	173
Australia . . . . .	185
Brazil . . . . .	251
Canada . . . . .	270
China . . . . .	295
CzechRepublic . . . . .	347
Denmark . . . . .	358
Finland . . . . .	437
Germany . . . . .	506
HongKong . . . . .	523
Hungary . . . . .	530
Iceland . . . . .	531
India . . . . .	550
Indonesia . . . . .	552
Italy . . . . .	581
Japan . . . . .	585
JointCalendar . . . . .	589
Mexico . . . . .	678
NewZealand . . . . .	704
Norway . . . . .	710
NullCalendar . . . . .	713
Poland . . . . .	760
SaudiArabia . . . . .	794
Singapore . . . . .	816
Slovakia . . . . .	824
SouthAfrica . . . . .	830
SouthKorea . . . . .	831
Sweden . . . . .	867
Switzerland . . . . .	868
Taiwan . . . . .	871
TARGET . . . . .	873
Turkey . . . . .	904
Ukraine . . . . .	910
UnitedKingdom . . . . .	912
UnitedStates . . . . .	914
Bridge< Constraint, ConstraintImpl > . . . . .	253
Constraint . . . . .	312
BoundaryConstraint . . . . .	248
CompositeConstraint . . . . .	306
NoConstraint . . . . .	706
PositiveConstraint . . . . .	761
Bridge< DayCounter, DayCounterImpl > . . . . .	253
DayCounter . . . . .	354
Actual360 . . . . .	151
Actual365Fixed . . . . .	152
ActualActual . . . . .	153
OneDayCounter . . . . .	731
SimpleDayCounter . . . . .	812
Thirty360 . . . . .	879
Bridge< Interpolation, InterpolationImpl > . . . . .	253
Interpolation . . . . .	568

BackwardFlatInterpolation . . . . .	188
CubicSpline . . . . .	336
MonotonicCubicSpline . . . . .	684
NaturalCubicSpline . . . . .	698
NaturalMonotonicCubicSpline . . . . .	699
ForwardFlatInterpolation . . . . .	456
LinearInterpolation . . . . .	622
LogLinearInterpolation . . . . .	639
Bridge< Interpolation2D, Interpolation2DImpl > . . . . .	253
Interpolation2D . . . . .	569
BicubicSpline . . . . .	207
BilinearInterpolation . . . . .	209
Bridge< Parameter, ParameterImpl > . . . . .	253
Parameter . . . . .	743
ConstantParameter . . . . .	311
NullParameter . . . . .	715
PiecewiseConstantParameter . . . . .	753
TermStructureFittingParameter . . . . .	877
ExtendedCoxIngersollRoss::FittingParameter . . . . .	422
G2::FittingParameter . . . . .	474
HullWhite::FittingParameter . . . . .	529
BrownianBridge . . . . .	255
CalendarImpl . . . . .	267
Calendar::OrthodoxImpl . . . . .	265
Calendar::WesternImpl . . . . .	266
Cashflows . . . . .	287
CLGaussianRng . . . . .	296
CliquetOption::arguments . . . . .	299
Composite . . . . .	305
ConstraintImpl . . . . .	313
ContinuousAveragingAsianOption::arguments . . . . .	316
ConvergenceStatistics . . . . .	318
ConvertibleBond::option::arguments . . . . .	319
ConvertibleFixedCouponBond . . . . .	321
ConvertibleFloatingRateBond . . . . .	322
ConvertibleZeroCouponBond . . . . .	323
CostFunction . . . . .	325
LeastSquareFunction . . . . .	605
CovarianceDecomposition . . . . .	328
CoxIngersollRoss::Dynamics . . . . .	331
ExtendedCoxIngersollRoss::Dynamics . . . . .	421
Cubic . . . . .	335
CumulativeBinomialDistribution . . . . .	338
CumulativeNormalDistribution . . . . .	339
CumulativePoissonDistribution . . . . .	340
CuriouslyRecurringTemplate . . . . .	341
Lattice . . . . .	598
Lattice1D . . . . .	600
Lattice2D . . . . .	601
Solver1D . . . . .	828
CuriouslyRecurringTemplate< AdditiveEQPBinoomialTree > . . . . .	341

Tree< AdditiveEQPBinoialTree > . . . . .	889
BinomialTree< AdditiveEQPBinoialTree > . . . . .	212
EqualProbabilitiesBinomialTree< AdditiveEQPBinoialTree > . . . . .	399
AdditiveEQPBinoialTree . . . . .	155
CuriouslyRecurringTemplate< Bisection > . . . . .	341
Solver1D< Bisection > . . . . .	828
Bisection . . . . .	214
CuriouslyRecurringTemplate< BlackScholesLattice< T > > . . . . .	341
Lattice< BlackScholesLattice< T > > . . . . .	598
Lattice1D< BlackScholesLattice< T > > . . . . .	600
BlackScholesLattice . . . . .	227
TsiveriotisFernandesLattice . . . . .	901
CuriouslyRecurringTemplate< Brent > . . . . .	341
Solver1D< Brent > . . . . .	828
Brent . . . . .	252
CuriouslyRecurringTemplate< CoxRossRubinstein > . . . . .	341
Tree< CoxRossRubinstein > . . . . .	889
BinomialTree< CoxRossRubinstein > . . . . .	212
EqualJumpsBinomialTree< CoxRossRubinstein > . . . . .	398
CoxRossRubinstein . . . . .	332
CuriouslyRecurringTemplate< FalsePosition > . . . . .	341
Solver1D< FalsePosition > . . . . .	828
FalsePosition . . . . .	427
CuriouslyRecurringTemplate< JarrowRudd > . . . . .	341
Tree< JarrowRudd > . . . . .	889
BinomialTree< JarrowRudd > . . . . .	212
EqualProbabilitiesBinomialTree< JarrowRudd > . . . . .	399
JarrowRudd . . . . .	587
CuriouslyRecurringTemplate< LeisenReimer > . . . . .	341
Tree< LeisenReimer > . . . . .	889
BinomialTree< LeisenReimer > . . . . .	212
LeisenReimer . . . . .	608
CuriouslyRecurringTemplate< Newton > . . . . .	341
Solver1D< Newton > . . . . .	828
Newton . . . . .	702
CuriouslyRecurringTemplate< NewtonSafe > . . . . .	341
Solver1D< NewtonSafe > . . . . .	828
NewtonSafe . . . . .	703
CuriouslyRecurringTemplate< OneFactorModel::ShortRateTree > . . . . .	341
Lattice< OneFactorModel::ShortRateTree > . . . . .	598
Lattice1D< OneFactorModel::ShortRateTree > . . . . .	600
OneFactorModel::ShortRateTree . . . . .	735
CuriouslyRecurringTemplate< Ridder > . . . . .	341
Solver1D< Ridder > . . . . .	828
Ridder . . . . .	785
CuriouslyRecurringTemplate< Secant > . . . . .	341
Solver1D< Secant > . . . . .	828
Secant . . . . .	796



CuriouslyRecurringTemplate< T > . . . . .	341
Tree . . . . .	889
BinomialTree . . . . .	212
EqualJumpsBinomialTree . . . . .	398
EqualProbabilitiesBinomialTree . . . . .	399
CuriouslyRecurringTemplate< Tian > . . . . .	341
Tree< Tian > . . . . .	889
BinomialTree< Tian > . . . . .	212
Tian . . . . .	880
CuriouslyRecurringTemplate< Trigeorgis > . . . . .	341
Tree< Trigeorgis > . . . . .	889
BinomialTree< Trigeorgis > . . . . .	212
EqualJumpsBinomialTree< Trigeorgis > . . . . .	398
Trigeorgis . . . . .	896
CuriouslyRecurringTemplate< TrinomialTree > . . . . .	341
Tree< TrinomialTree > . . . . .	889
TrinomialTree . . . . .	897
CuriouslyRecurringTemplate< TwoFactorModel::ShortRateTree > . . . . .	341
Lattice< TwoFactorModel::ShortRateTree > . . . . .	598
Lattice2D< TwoFactorModel::ShortRateTree, TrinomialTree > . . . . .	601
TwoFactorModel::ShortRateTree . . . . .	908
Currency . . . . .	342
ARSCurrency . . . . .	180
ATSCurrency . . . . .	182
AUDCurrency . . . . .	183
BDTCurrency . . . . .	202
BEFCurrency . . . . .	203
BGLCurrency . . . . .	205
BRLCurrency . . . . .	254
BYRCurrency . . . . .	257
CADCurrency . . . . .	258
CHFCurrency . . . . .	293
CLPCurrency . . . . .	302
CNYCurrency . . . . .	303
COPCurrency . . . . .	324
CYPCurrency . . . . .	346
CZKCurrency . . . . .	349
DEMCurrency . . . . .	357
DKKCurrency . . . . .	384
EEKCurrency . . . . .	395
ESPCurrency . . . . .	402
EURCurrency . . . . .	405
FIMCurrency . . . . .	435
FRFCurrency . . . . .	470
GBPCurrency . . . . .	495
GRDCurrency . . . . .	509
HKDCurrency . . . . .	522
HUFCurrency . . . . .	525
IEPCurrency . . . . .	533
ILSCurrency . . . . .	534
INRCurrency . . . . .	554

IQDCurrency . . . . .	578
IRRCurrency . . . . .	579
ISKCurrency . . . . .	580
ITLCurrency . . . . .	583
JPYCurrency . . . . .	590
KRWCurrency . . . . .	596
KWDCurrency . . . . .	597
LTLCurrency . . . . .	640
LUFCurrency . . . . .	641
LVLCurrency . . . . .	642
MTLCurrency . . . . .	688
MXNCurrency . . . . .	697
NLGCurrency . . . . .	705
NOKCurrency . . . . .	707
NPRCurrency . . . . .	711
NZDCurrency . . . . .	718
PKRCurrency . . . . .	756
PLNCurrency . . . . .	758
PTECurrency . . . . .	767
ROLCurrency . . . . .	786
SARCurrency . . . . .	793
SEKCurrency . . . . .	799
SGDCurrency . . . . .	804
SITCurrency . . . . .	822
SKKCurrency . . . . .	823
THBCurrency . . . . .	878
TRLCurrency . . . . .	898
TRYCurrency . . . . .	900
TTDCurrency . . . . .	903
TWDCurrency . . . . .	905
USDCurrency . . . . .	920
VEBCurrency . . . . .	932
ZARCurrency . . . . .	940
Date . . . . .	350
DayCounterImpl . . . . .	356
Discount . . . . .	364
DiscreteAveragingAsianOption::arguments . . . . .	368
DiscretizedAsset . . . . .	371
DiscretizedDiscountBond . . . . .	374
DiscretizedOption . . . . .	375
Disposable . . . . .	377
DividendVanillaOption::arguments . . . . .	382
Duration . . . . .	392
EndCriteria . . . . .	396
ErrorFunction . . . . .	401
exception . . . . .	
Error . . . . .	400
ExchangeRate . . . . .	412
Exercise . . . . .	416
EarlyExercise . . . . .	394
AmericanExercise . . . . .	160
BermudanExercise . . . . .	204
EuropeanExercise . . . . .	408

Extrapolator . . . . .	425
BlackVolTermStructure . . . . .	239
BlackVarianceTermStructure . . . . .	235
BlackVarianceCurve . . . . .	231
BlackVarianceSurface . . . . .	233
ImpliedVolTermStructure . . . . .	539
BlackVolatilityTermStructure . . . . .	237
BlackConstantVol . . . . .	219
CapletVolatilityStructure . . . . .	279
CapletConstantVolatility . . . . .	277
CapVolatilityStructure . . . . .	281
CapVolatilityVector . . . . .	283
Interpolation . . . . .	568
Interpolation2D . . . . .	569
LocalVolTermStructure . . . . .	636
LocalConstantVol . . . . .	631
LocalVolCurve . . . . .	632
LocalVolSurface . . . . .	634
SwaptionVolatilityStructure . . . . .	865
SwaptionVolatilityMatrix . . . . .	863
YieldTermStructure . . . . .	936
AffineTermStructure . . . . .	157
FlatForward . . . . .	446
ForwardRateStructure . . . . .	460
CompoundForward . . . . .	308
ForwardSpreadedTermStructure . . . . .	462
InterpolatedForwardCurve . . . . .	564
ImpliedTermStructure . . . . .	537
InterpolatedDiscountCurve . . . . .	562
ExtendedDiscountCurve . . . . .	423
InterpolatedDiscountCurve< LogLinear > . . . . .	562
ZeroYieldStructure . . . . .	946
DriftTermStructure . . . . .	390
InterpolatedZeroCurve . . . . .	566
QuantoTermStructure . . . . .	774
ZeroSpreadedTermStructure . . . . .	943
Factorial . . . . .	426
FaureRsg . . . . .	428
FDAmericanCondition . . . . .	429
FDDividendEngineMerton73 . . . . .	431
FDDividendEngineShiftScale . . . . .	432
FDEuropeanEngine . . . . .	433
FDStepConditionEngine . . . . .	434
FiniteDifferenceModel . . . . .	436
ForwardFlat . . . . .	455
ForwardOptionArguments . . . . .	457
ForwardRate . . . . .	459
GammaFunction . . . . .	476
GaussianOrthogonalPolynomial . . . . .	485
GaussHermitePolynomial . . . . .	482
GaussHyperbolicPolynomial . . . . .	484
GaussJacobiPolynomial . . . . .	491

GaussLaguerrePolynomial . . . . .	493
GaussianQuadrature . . . . .	486
GaussChebyshev2thIntegration . . . . .	478
GaussChebyshevIntegration . . . . .	479
GaussGegenbauerIntegration . . . . .	480
GaussHermiteIntegration . . . . .	481
GaussHyperbolicIntegration . . . . .	483
GaussJacobiIntegration . . . . .	490
GaussLaguerreIntegration . . . . .	492
GaussLegendreIntegration . . . . .	494
GaussianStatistics . . . . .	487
GeneralStatistics . . . . .	497
GenericRiskStatistics . . . . .	502
HaltonRsg . . . . .	511
Handle . . . . .	512
History . . . . .	517
History::const_iterator . . . . .	520
History::Entry . . . . .	521
HullWhite::Dynamics . . . . .	528
IMM . . . . .	535
IncrementalStatistics . . . . .	543
IntegralEngine . . . . .	558
InterestRate . . . . .	559
Interpolation2DImpl . . . . .	571
Interpolation2D::templateImpl . . . . .	570
InterpolationImpl . . . . .	573
Interpolation::templateImpl . . . . .	572
InverseCumulativeNormal . . . . .	574
InverseCumulativePoisson . . . . .	575
InverseCumulativeRng . . . . .	576
InverseCumulativeRsg . . . . .	577
KnuthUniformRng . . . . .	594
KronrodIntegral . . . . .	595
LeastSquareProblem . . . . .	606
LecuyerUniformRng . . . . .	607
LexicographicalView . . . . .	610
LfmCovarianceParameterization . . . . .	612
LfmCovarianceProxy . . . . .	613
LfmHullWhiteParameterization . . . . .	614
Linear . . . . .	621
LineSearch . . . . .	623
ArmijoLineSearch . . . . .	176
LmCorrelationModel . . . . .	627
LmExponentialCorrelationModel . . . . .	628
LmVolatilityModel . . . . .	630
LmLinearExponentialVolatilityModel . . . . .	629
LogLinear . . . . .	638
MakeMCDigitalEngine . . . . .	643
MakeMCEuropeanEngine . . . . .	644
MakeMCEuropeanHestonEngine . . . . .	645
MakeSchedule . . . . .	646

map< Date, Real >	
TimeBasket . . . . .	882
Matrix . . . . .	647
McPricer . . . . .	671
McPricer< MultiVariate< PseudoRandom > > . . . . .	671
McEverest . . . . .	666
McHimalaya . . . . .	667
McMaxBasket . . . . .	668
McPagoda . . . . .	669
McPricer< SingleVariate< PseudoRandom > > . . . . .	671
McCliquetOption . . . . .	656
McDiscreteArithmeticASO . . . . .	660
McPerformanceOption . . . . .	670
McSimulation . . . . .	672
MCVanillaEngine . . . . .	674
McSimulation< MultiVariate< RNG >, S > . . . . .	672
MCBasketEngine . . . . .	654
MCVanillaEngine< MultiVariate< RNG >, S > . . . . .	674
MCEuropeanHestonEngine . . . . .	665
McSimulation< SingleVariate< RNG >, S > . . . . .	672
MCBarrierEngine . . . . .	652
MCDiscreteAveragingAsianEngine . . . . .	661
MCDiscreteArithmeticAPEngine . . . . .	658
MCDiscreteGeometricAPEngine . . . . .	663
MCVanillaEngine< SingleVariate< RNG >, S > . . . . .	674
MCDigitalEngine . . . . .	657
MCEuropeanEngine . . . . .	664
MersenneTwisterUniformRng . . . . .	675
MixedScheme . . . . .	680
CrankNicolson . . . . .	333
ExplicitEuler . . . . .	417
ImplicitEuler . . . . .	536
Money . . . . .	682
MonteCarloModel . . . . .	685
MoroInverseCumulativeNormal . . . . .	687
MultiCubicSpline . . . . .	693
MultiPath . . . . .	694
MultiPathGenerator . . . . .	695
MultiVariate . . . . .	696
NonLinearLeastSquare . . . . .	708
NormalDistribution . . . . .	709
Null . . . . .	712
NumericalMethod . . . . .	716
Lattice . . . . .	598
Lattice< BlackScholesLattice< T > > . . . . .	598
Lattice< OneFactorModel::ShortRateTree > . . . . .	598
Lattice< TwoFactorModel::ShortRateTree > . . . . .	598
Observable . . . . .	720
AffineModel . . . . .	156
G2 . . . . .	472
LiborForwardModel . . . . .	617

OneFactorAffineModel . . . . .	732
CoxIngersollRoss . . . . .	329
ExtendedCoxIngersollRoss . . . . .	419
Vasicek . . . . .	929
HullWhite . . . . .	526
BlackModel . . . . .	225
CalibrationHelper . . . . .	268
CapHelper . . . . .	276
HestonModelHelper . . . . .	514
SwaptionHelper . . . . .	862
Event . . . . .	410
CashFlow . . . . .	285
Coupon . . . . .	326
FixedRateCoupon . . . . .	444
FloatingRateCoupon . . . . .	450
IndexedCoupon . . . . .	547
InArrearIndexedCoupon . . . . .	541
UpFrontIndexedCoupon . . . . .	917
ParCoupon . . . . .	745
Short< ParCoupon > . . . . .	806
Dividend . . . . .	378
FixedDividend . . . . .	442
FractionalDividend . . . . .	466
SimpleCashFlow . . . . .	810
Index . . . . .	546
Xibor . . . . .	934
Cdor . . . . .	291
Euribor . . . . .	406
Jibar . . . . .	588
Libor . . . . .	616
AUDLibor . . . . .	184
CADLibor . . . . .	259
CHFLibor . . . . .	294
DKKLibor . . . . .	385
EURLibor . . . . .	407
GBPLibor . . . . .	496
JPYLibor . . . . .	591
NZDLibor . . . . .	719
USDLibor . . . . .	921
Tibor . . . . .	881
TRLibor . . . . .	899
Zibor . . . . .	948
LazyObject . . . . .	603
AffineTermStructure . . . . .	157
Instrument . . . . .	555
Bond . . . . .	242
FixedCouponBond . . . . .	438
FloatingRateBond . . . . .	448
ZeroCouponBond . . . . .	942
CapFloor . . . . .	272
Cap . . . . .	271
Collar . . . . .	304

Floor . . . . .	452
Option . . . . .	739
MultiAssetOption . . . . .	689
BasketOption . . . . .	195
OneAssetOption . . . . .	724
OneAssetStrikedOption . . . . .	729
BarrierOption . . . . .	191
CliquetOption . . . . .	297
ContinuousAveragingAsianOption . . . . .	314
DiscreteAveragingAsianOption . . . . .	366
VanillaOption . . . . .	923
DividendVanillaOption . . . . .	380
EuropeanOption . . . . .	409
ForwardVanillaOption . . . . .	464
QuantoVanillaOption . . . . .	776
QuantoForwardVanillaOption . . . . .	770
Swaption . . . . .	858
Stock . . . . .	848
Swap . . . . .	853
VanillaSwap . . . . .	925
PiecewiseYieldCurve . . . . .	754
Link . . . . .	625
PricingEngine . . . . .	763
GenericEngine . . . . .	500
GenericModelEngine . . . . .	501
GenericEngine< Arguments, Results > . . . . .	500
GenericModelEngine< ShortRateModel, Arguments, Results > . . . . .	501
LatticeShortRateModelEngine . . . . .	602
GenericEngine< BarrierOption::arguments, BarrierOption::results > . . . . .	500
BarrierOption::engine . . . . .	194
AnalyticBarrierEngine . . . . .	163
MCBarrierEngine . . . . .	652
GenericEngine< BasketOption::arguments, BasketOption::results > . . . . .	500
BasketOption::engine . . . . .	198
MCAmericanBasketEngine . . . . .	651
MCBasketEngine . . . . .	654
StulzEngine . . . . .	850
GenericEngine< CapFloor::arguments, CapFloor::results > . . . . .	500
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results > . . . . .	501
AnalyticCapFloorEngine . . . . .	164
GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results > . . . . .	501
BlackCapFloorEngine . . . . .	218
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results > . . . . .	501
LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results > . . . . .	602
TreeCapFloorEngine . . . . .	890
GenericEngine< CliquetOption::arguments, CliquetOption::results > . . . . .	500
CliquetOption::engine . . . . .	300
AnalyticCliquetEngine . . . . .	165
AnalyticPerformanceEngine . . . . .	172
GenericEngine< ContinuousAveragingAsianOption::arguments, ContinuousAveragingAsianOption::results > . . . . .	500

ContinuousAveragingAsianOption::engine	317
AnalyticContinuousGeometricAveragePriceAsianEngine	166
GenericEngine< ConvertibleBond::option::arguments, ConvertibleBond::option::results >	500
ConvertibleBond::option::engine	320
BinomialConvertibleEngine	210
GenericEngine< DiscreteAveragingAsianOption::arguments, DiscreteAveragingAsianOption::results >	500
DiscreteAveragingAsianOption::engine	369
AnalyticDiscreteGeometricAveragePriceAsianEngine	168
MCDiscreteAveragingAsianEngine	661
GenericEngine< DividendVanillaOption::arguments, DividendVanillaOption::results >	500
DividendVanillaOption::engine	383
AnalyticDividendEuropeanEngine	169
GenericEngine< ForwardOptionArguments< ArgumentsType >, ResultsType >	500
ForwardEngine	454
ForwardPerformanceEngine	458
GenericEngine< OneAssetOption::arguments, OneAssetOption::results >	500
GenericEngine< QuantoOptionArguments< ArgumentsType >, QuantoOptionResults< ResultsType > >	500
QuantoEngine	768
GenericEngine< Swaption::arguments, Swaption::results >	500
GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results >	501
BlackSwaptionEngine	230
GenericModelEngine< G2, Swaption::arguments, Swaption::results >	501
G2SwaptionEngine	475
GenericModelEngine< LiborForwardModel, Swaption::arguments, Swaption::results >	501
LfmSwaptionEngine	615
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results >	501
JamshidianSwaptionEngine	584
GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results >	501
LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >	602
TreeSwaptionEngine	891
GenericEngine< VanillaOption::arguments, VanillaOption::results >	500
GenericModelEngine< HestonModel, VanillaOption::arguments, VanillaOption::results >	501
AnalyticHestonEngine	171
BatesEngine	199
GenericEngine< VanillaSwap::arguments, VanillaSwap::results >	500
GenericModelEngine< ShortRateModel, VanillaSwap::arguments, VanillaSwap::results >	501
LatticeShortRateModelEngine< VanillaSwap::arguments, VanillaSwap::results >	602
TreeVanillaSwapEngine	892
Quote	778
CompositeQuote	307
DerivedQuote	361



SimpleQuote . . . . .	813
RateHelper . . . . .	782
DepositRateHelper . . . . .	359
FixedCouponBondHelper . . . . .	440
FraRateHelper . . . . .	468
FuturesRateHelper . . . . .	471
SwapRateHelper . . . . .	855
ShortRateModel . . . . .	807
HestonModel . . . . .	513
BatesModel . . . . .	201
LiborForwardModel . . . . .	617
OneFactorModel . . . . .	733
BlackKarasinski . . . . .	223
OneFactorAffineModel . . . . .	732
TwoFactorModel . . . . .	906
G2 . . . . .	472
StochasticProcess . . . . .	839
HestonProcess . . . . .	515
LiborForwardModelProcess . . . . .	619
StochasticProcess1D . . . . .	842
BlackScholesProcess . . . . .	228
GeometricBrownianMotionProcess . . . . .	505
Merton76Process . . . . .	676
OrnsteinUhlenbeckProcess . . . . .	741
SquareRootProcess . . . . .	833
StochasticProcessArray . . . . .	846
TermStructure . . . . .	874
BlackVolTermStructure . . . . .	239
CapletVolatilityStructure . . . . .	279
CapVolatilityStructure . . . . .	281
LocalVolTermStructure . . . . .	636
SwaptionVolatilityStructure . . . . .	865
YieldTermStructure . . . . .	936
TermStructureConsistentModel . . . . .	876
BlackKarasinski . . . . .	223
ExtendedCoxIngersollRoss . . . . .	419
G2 . . . . .	472
HullWhite . . . . .	526
ObservableValue . . . . .	721
ObservableValue< Date > . . . . .	721
Observer . . . . .	722
BlackModel . . . . .	225
CalibrationHelper . . . . .	268
CompositeQuote . . . . .	307
DerivedQuote . . . . .	361
GenericModelEngine . . . . .	501
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results > . . .	501
GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results > . . .	501
GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results > . . .	501
GenericModelEngine< G2, Swaption::arguments, Swaption::results > . . . . .	501
GenericModelEngine< HestonModel, VanillaOption::arguments, Vanilla- Option::results > . . . . .	501

GenericModelEngine< LiborForwardModel, Swaption::arguments, Swaption::results > . . . . .	501
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results > . . . . .	501
GenericModelEngine< ShortRateModel, Arguments, Results > . . . . .	501
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results > . . . . .	501
GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results > . . . . .	501
GenericModelEngine< ShortRateModel, VanillaSwap::arguments, VanillaSwap::results > . . . . .	501
IndexedCoupon . . . . .	547
LazyObject . . . . .	603
Link . . . . .	625
ParCoupon . . . . .	745
RateHelper . . . . .	782
ShortRateModel . . . . .	807
StochasticProcess . . . . .	839
TermStructure . . . . .	874
Xibor . . . . .	934
OneFactorModel::ShortRateDynamics . . . . .	734
OperatorFactory . . . . .	736
OptimizationMethod . . . . .	737
ConjugateGradient . . . . .	310
LevenbergMarquardt . . . . .	609
Simplex . . . . .	814
SteepestDescent . . . . .	835
ParameterImpl . . . . .	744
Path . . . . .	747
PathGenerator . . . . .	748
PathPricer . . . . .	749
PathPricer< MultiPath > . . . . .	749
PathPricer< Path > . . . . .	749
Payoff . . . . .	750
TypePayoff . . . . .	909
StrikedTypePayoff . . . . .	849
AssetOrNothingPayoff . . . . .	181
CashOrNothingPayoff . . . . .	290
GapPayoff . . . . .	477
PercentageStrikePayoff . . . . .	751
PlainVanillaPayoff . . . . .	757
SuperSharePayoff . . . . .	851
Period . . . . .	752
PoissonDistribution . . . . .	759
PrimeNumbers . . . . .	764
Problem . . . . .	765
QuantoOptionArguments . . . . .	772
QuantoOptionResults . . . . .	773
RandomizedLDS . . . . .	779
RandomSequenceGenerator . . . . .	781
Results . . . . .	784
Greeks . . . . .	510
MultiAssetOption::results . . . . .	692
OneAssetOption::results . . . . .	728
MoreGreeks . . . . .	686

OneAssetOption::results . . . . .	728
PriceCurve . . . . .	762
OneAssetOption::results . . . . .	728
Value . . . . .	922
CapFloor::results . . . . .	275
MultiAssetOption::results . . . . .	692
OneAssetOption::results . . . . .	728
Swaption::results . . . . .	861
VanillaSwap::results . . . . .	928
Rounding . . . . .	787
CeilingTruncation . . . . .	292
ClosestRounding . . . . .	301
DownRounding . . . . .	387
FloorTruncation . . . . .	453
UpRounding . . . . .	919
SalvagingAlgorithm . . . . .	789
Sample . . . . .	790
SampledCurve . . . . .	791
Schedule . . . . .	795
SegmentIntegral . . . . .	798
SequenceStatistics . . . . .	800
SequenceStatistics< Statistics > . . . . .	800
DiscrepancyStatistics . . . . .	365
Short . . . . .	805
ShoutCondition . . . . .	809
SingleAssetOption . . . . .	818
DiscreteGeometricASO . . . . .	370
Singleton . . . . .	820
ExchangeRateManager . . . . .	414
IndexManager . . . . .	549
SeedGenerator . . . . .	797
Settings . . . . .	802
Singleton< ExchangeRateManager > . . . . .	820
Singleton< IndexManager > . . . . .	820
Singleton< SeedGenerator > . . . . .	820
Singleton< Settings > . . . . .	820
Singleton< Tracing > . . . . .	820
SingleVariate . . . . .	821
SobolRsg . . . . .	826
StatsHolder . . . . .	834
step_iterator . . . . .	836
StepCondition . . . . .	837
NullCondition . . . . .	714
ZeroCondition . . . . .	941
StepConditionSet . . . . .	838
StochasticProcess1D::discretization . . . . .	844
EulerDiscretization . . . . .	403
StochasticProcess::discretization . . . . .	845
EulerDiscretization . . . . .	403
SVD . . . . .	852
SymmetricSchurDecomposition . . . . .	869

TabulatedGaussLegendre . . . . .	870
TimeGrid . . . . .	883
TqrEigenDecomposition . . . . .	885
TransformedGrid . . . . .	886
TrapezoidIntegral . . . . .	887
SimpsonIntegral . . . . .	815
TridiagonalOperator . . . . .	893
BSMOperator . . . . .	256
DMinus . . . . .	386
DPlus . . . . .	388
DPlusDMinus . . . . .	389
DZero . . . . .	393
TridiagonalOperator::TimeSetter . . . . .	895
TwoFactorModel::ShortRateDynamics . . . . .	907
VanillaOption::engine . . . . .	924
AnalyticDigitalAmericanEngine . . . . .	167
AnalyticEuropeanEngine . . . . .	170
BaroneAdesiWhaleyApproximationEngine . . . . .	189
BinomialVanillaEngine . . . . .	213
BjerkstrandStenslandApproximationEngine . . . . .	217
FDBermudanEngine . . . . .	430
JumpDiffusionEngine . . . . .	592
JuQuadraticApproximationEngine . . . . .	593
MCVanillaEngine . . . . .	674
MCVanillaEngine< MultiVariate< RNG >, S > . . . . .	674
MCVanillaEngine< SingleVariate< RNG >, S > . . . . .	674
Vasicek::Dynamics . . . . .	931
Visitor . . . . .	933
Visitor< CashFlow > . . . . .	933
BPSBasketCalculator . . . . .	250
Visitor< Coupon > . . . . .	933
BPSBasketCalculator . . . . .	250
Visitor< FixedRateCoupon > . . . . .	933
BPSBasketCalculator . . . . .	250
ZeroYield . . . . .	945

## Chapter 4

# QuantLib Class Index

### 4.1 QuantLib Class List

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

<a href="#">Actual360</a> (Actual/360 day count convention ) . . . . .	151
<a href="#">Actual365Fixed</a> (Actual/365 (Fixed) day count convention ) . . . . .	152
<a href="#">ActualActual</a> (Actual/Actual day count ) . . . . .	153
<a href="#">AcyclicVisitor</a> (Degenerate base class for the Acyclic Visitor pattern ) . . . . .	154
<a href="#">AdditiveEQPBinomialTree</a> (Additive equal probabilities binomial tree ) . . . . .	155
<a href="#">AffineModel</a> (Affine model class ) . . . . .	156
<a href="#">AffineTermStructure</a> (Term-structure implied by an affine model ) . . . . .	157
<a href="#">AmericanCondition</a> (American exercise condition ) . . . . .	159
<a href="#">AmericanExercise</a> (American exercise ) . . . . .	160
<a href="#">AmericanPayoffAtExpiry</a> . . . . .	161
<a href="#">AmericanPayoffAtHit</a> . . . . .	162
<a href="#">AnalyticBarrierEngine</a> (Pricing engine for barrier options using analytical formulae ) . .	163
<a href="#">AnalyticCapFloorEngine</a> (Analytic engine for cap/floor ) . . . . .	164
<a href="#">AnalyticCliquetEngine</a> (Pricing engine for Cliquet options using analytical formulae ) .	165
<a href="#">AnalyticContinuousGeometricAveragePriceAsianEngine</a> (Pricing engine for European continuous geometric average price Asian ) . . . . .	166
<a href="#">AnalyticDigitalAmericanEngine</a> . . . . .	167
<a href="#">AnalyticDiscreteGeometricAveragePriceAsianEngine</a> (Pricing engine for European dis- crete geometric average price Asian ) . . . . .	168
<a href="#">AnalyticDividendEuropeanEngine</a> (Analytic pricing engine for European options with discrete dividends ) . . . . .	169
<a href="#">AnalyticEuropeanEngine</a> (Pricing engine for European vanilla options using analytical formulae ) . . . . .	170
<a href="#">AnalyticHestonEngine</a> (Analytic Heston-model engine based on Fourier transform ) . .	171
<a href="#">AnalyticPerformanceEngine</a> (Pricing engine for performance options using analytical formulae ) . . . . .	172
<a href="#">Argentina</a> (Argentinian calendars ) . . . . .	173
<a href="#">Arguments</a> (Base class for generic argument groups ) . . . . .	175
<a href="#">ArmijoLineSearch</a> (Armijo line search ) . . . . .	176
<a href="#">Array</a> (1-D array used in linear algebra ) . . . . .	177
<a href="#">ARSCurrency</a> (Argentinian peso ) . . . . .	180
<a href="#">AssetOrNothingPayoff</a> (Binary asset-or-nothing payoff ) . . . . .	181
<a href="#">ATSCurrency</a> (Austrian shilling ) . . . . .	182

<a href="#">AUDCurrency</a> (Australian dollar ) . . . . .	183
<a href="#">AUDLibor</a> (AUD LIBOR rate ) . . . . .	184
<a href="#">Australia</a> (Australian calendar ) . . . . .	185
<a href="#">Average</a> (Placeholder for enumerated averaging types ) . . . . .	186
<a href="#">BackwardFlat</a> (Backward-flat interpolation factory and traits ) . . . . .	187
<a href="#">BackwardFlatInterpolation</a> (Backward-flat interpolation between discrete points ) . . . .	188
<a href="#">BaroneAdesiWhaleyApproximationEngine</a> . . . . .	189
<a href="#">Barrier</a> (Placeholder for enumerated barrier types ) . . . . .	190
<a href="#">BarrierOption</a> (Barrier option on a single asset ) . . . . .	191
<a href="#">BarrierOption::arguments</a> (Arguments for barrier option calculation ) . . . . .	193
<a href="#">BarrierOption::engine</a> (Barrier engine base class ) . . . . .	194
<a href="#">BasketOption</a> (Basket option on a number of assets ) . . . . .	195
<a href="#">BasketOption::arguments</a> (Arguments for basket option calculation ) . . . . .	197
<a href="#">BasketOption::engine</a> (Basket option engine base class ) . . . . .	198
<a href="#">BatesEngine</a> (Bates model engines based on Fourier transform ) . . . . .	199
<a href="#">BatesModel</a> . . . . .	201
<a href="#">BDTCurrency</a> (Bangladesh taka ) . . . . .	202
<a href="#">BEFCurrency</a> (Belgian franc ) . . . . .	203
<a href="#">BermudanExercise</a> (Bermudan exercise ) . . . . .	204
<a href="#">BGLCurrency</a> (Bulgarian lev ) . . . . .	205
<a href="#">Bicubic</a> (Bicubic-spline interpolation factory ) . . . . .	206
<a href="#">BicubicSpline</a> . . . . .	207
<a href="#">Bilinear</a> ( <a href="#">Bilinear</a> interpolation factory ) . . . . .	208
<a href="#">BilinearInterpolation</a> ( <a href="#">Bilinear</a> interpolation between discrete points ) . . . . .	209
<a href="#">BinomialConvertibleEngine</a> (Binomial Tsiveriotis-Fernandes engine for convertible bonds ) . . . . .	210
<a href="#">BinomialDistribution</a> (Binomial probability distribution function ) . . . . .	211
<a href="#">BinomialTree</a> (Binomial tree base class ) . . . . .	212
<a href="#">BinomialVanillaEngine</a> (Pricing engine for vanilla options using binomial trees ) . . . .	213
<a href="#">Bisection</a> (Bisection 1-D solver ) . . . . .	214
<a href="#">BivariateCumulativeNormalDistributionDr78</a> (Cumulative bivariate normal distribu- tion function ) . . . . .	215
<a href="#">BivariateCumulativeNormalDistributionWe04DP</a> (Cumulative bivariate normal distri- bution function (West 2004) ) . . . . .	216
<a href="#">BjerkstrandStenslandApproximationEngine</a> . . . . .	217
<a href="#">BlackCapFloorEngine</a> (Black-formula cap/floor engine ) . . . . .	218
<a href="#">BlackConstantVol</a> (Constant Black volatility, no time-strike dependence ) . . . . .	219
<a href="#">BlackFormula</a> (Black-formula calculator ) . . . . .	221
<a href="#">BlackKarasinski</a> (Standard Black-Karasinski model class ) . . . . .	223
<a href="#">BlackKarasinski::Dynamics</a> (Short-rate dynamics in the Black-Karasinski model ) . . . .	224
<a href="#">BlackModel</a> (Black-model for vanilla interest-rate derivatives ) . . . . .	225
<a href="#">BlackScholesLattice</a> (Simple binomial lattice approximating the Black-Scholes model ) .	227
<a href="#">BlackScholesProcess</a> (Black-Scholes stochastic process ) . . . . .	228
<a href="#">BlackSwaptionEngine</a> (Black-formula swaption engine ) . . . . .	230
<a href="#">BlackVarianceCurve</a> (Black volatility curve modelled as variance curve ) . . . . .	231
<a href="#">BlackVarianceSurface</a> (Black volatility surface modelled as variance surface ) . . . . .	233
<a href="#">BlackVarianceTermStructure</a> (Black variance term structure ) . . . . .	235
<a href="#">BlackVolatilityTermStructure</a> (Black-volatility term structure ) . . . . .	237
<a href="#">BlackVolTermStructure</a> (Black-volatility term structure ) . . . . .	239
<a href="#">Bond</a> (Base bond class ) . . . . .	242
<a href="#">BoundaryCondition</a> (Abstract boundary condition class for finite difference problems )	246
<a href="#">BoundaryConstraint</a> (Constraint imposing all arguments to be in [low,high] ) . . . . .	248
<a href="#">BoxMullerGaussianRng</a> (Gaussian random number generator ) . . . . .	249
<a href="#">BPSBasketCalculator</a> . . . . .	250

Brazil (Brazilian calendar ) . . . . .	251
Brent (Brent 1-D solver ) . . . . .	252
Bridge (The Bridge pattern made explicit ) . . . . .	253
BRLCurrency (Brazilian real ) . . . . .	254
BrownianBridge (Builds Wiener process paths using Gaussian variates ) . . . . .	255
BSMOperator (Black-Scholes-Merton differential operator ) . . . . .	256
BYRCurrency (Belarussian ruble ) . . . . .	257
CADCurrency (Canadian dollar ) . . . . .	258
CADLibor (CAD LIBOR rate ) . . . . .	259
Calendar (calendar class ) . . . . .	260
Calendar::OrthodoxImpl (Partial calendar implementation ) . . . . .	265
Calendar::WesternImpl (Partial calendar implementation ) . . . . .	266
CalendarImpl (Abstract base class for calendar implementations ) . . . . .	267
CalibrationHelper (Liquid market instrument used during calibration ) . . . . .	268
Canada (Canadian calendar ) . . . . .	270
Cap (Concrete cap class ) . . . . .	271
CapFloor (Base class for cap-like instruments ) . . . . .	272
CapFloor::arguments (Arguments for cap/floor calculation ) . . . . .	274
CapFloor::results (Results from cap/floor calculation ) . . . . .	275
CapHelper (Calibration helper for ATM cap ) . . . . .	276
CapletConstantVolatility (Constant caplet volatility, no time-strike dependence ) . . . . .	277
CapletVolatilityStructure (Caplet/floorlet forward-volatility structure ) . . . . .	279
CapVolatilityStructure (Cap/floor term-volatility structure ) . . . . .	281
CapVolatilityVector (Cap/floor at-the-money term-volatility vector ) . . . . .	283
CashFlow (Base class for cash flows ) . . . . .	285
Cashflows (Cashflows analysis functions ) . . . . .	287
CashOrNothingPayoff (Binary cash-or-nothing payoff ) . . . . .	290
Cdor (CDOR rate ) . . . . .	291
CeilingTruncation (Ceiling truncation ) . . . . .	292
CHFCurrency (Swiss franc ) . . . . .	293
CHFLibor (CHF LIBOR rate ) . . . . .	294
China (Chinese calendar ) . . . . .	295
CLGaussianRng (Gaussian random number generator ) . . . . .	296
CliquetOption (Cliquet (Ratchet) option ) . . . . .	297
CliquetOption::arguments (Arguments for cliquet option calculation ) . . . . .	299
CliquetOption::engine (Cliquet engine base class ) . . . . .	300
ClosestRounding (Closest rounding ) . . . . .	301
CLPCurrency (Chilean peso ) . . . . .	302
CNYCurrency (Chinese yuan ) . . . . .	303
Collar (Concrete collar class ) . . . . .	304
Composite (Composite pattern ) . . . . .	305
CompositeConstraint (Constraint enforcing both given sub-constraints ) . . . . .	306
CompositeQuote (Market element whose value depends on two other market element ) . . . . .	307
CompoundForward (Compound-forward structure ) . . . . .	308
ConjugateGradient (Multi-dimensional Conjugate Gradient class ) . . . . .	310
ConstantParameter (Standard constant parameter $a(t) = a$ ) . . . . .	311
Constraint (Base constraint class ) . . . . .	312
ConstraintImpl (Base class for constraint implementations ) . . . . .	313
ContinuousAveragingAsianOption (Continuous-averaging Asian option ) . . . . .	314
ContinuousAveragingAsianOption::arguments (Extra arguments for single-asset continuous-average Asian option ) . . . . .	316
ContinuousAveragingAsianOption::engine (Continuous-averaging Asian engine base class ) . . . . .	317
ConvergenceStatistics (Statistics class with convergence table ) . . . . .	318



<a href="#">ConvertibleBond::option::arguments</a> (Arguments for Convertible <a href="#">Bond</a> calculation ) . .	319
<a href="#">ConvertibleBond::option::engine</a> (Convertible bond engine base class ) . . . . .	320
<a href="#">ConvertibleFixedCouponBond</a> (Convertible fixed-coupon bond ) . . . . .	321
<a href="#">ConvertibleFloatingRateBond</a> (Convertible floating-rate bond ) . . . . .	322
<a href="#">ConvertibleZeroCouponBond</a> (Convertible zero-coupon bond ) . . . . .	323
<a href="#">COPCurrency</a> (Colombian peso ) . . . . .	324
<a href="#">CostFunction</a> (Cost function abstract class for optimization problem ) . . . . .	325
<a href="#">Coupon</a> (coupon accruing over a fixed period ) . . . . .	326
<a href="#">CovarianceDecomposition</a> . . . . .	328
<a href="#">CoxIngersollRoss</a> (Cox-Ingersoll-Ross model class ) . . . . .	329
<a href="#">CoxIngersollRoss::Dynamics</a> (Dynamics of the short-rate under the Cox-Ingersoll-Ross model ) . . . . .	331
<a href="#">CoxRossRubinstein</a> (Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree ) .	332
<a href="#">CrankNicolson</a> (Crank-Nicolson scheme for finite difference methods ) . . . . .	333
<a href="#">Cubic</a> (cubic-spline interpolation factory and traits ) . . . . .	335
<a href="#">CubicSpline</a> (Cubic spline interpolation between discrete points ) . . . . .	336
<a href="#">CumulativeBinomialDistribution</a> (Cumulative binomial distribution function ) . . . .	338
<a href="#">CumulativeNormalDistribution</a> (Cumulative normal distribution function ) . . . . .	339
<a href="#">CumulativePoissonDistribution</a> (Cumulative Poisson distribution function ) . . . . .	340
<a href="#">CuriouslyRecurringTemplate</a> (Support for the curiously recurring template pattern ) . .	341
<a href="#">Currency</a> (Currency specification ) . . . . .	342
<a href="#">CYPCurrency</a> (Cyprus pound ) . . . . .	346
<a href="#">CzechRepublic</a> (Czech calendars ) . . . . .	347
<a href="#">CZKCurrency</a> (Czech koruna ) . . . . .	349
<a href="#">Date</a> (Concrete date class ) . . . . .	350
<a href="#">DayCounter</a> (Day counter class ) . . . . .	354
<a href="#">DayCounterImpl</a> (Abstract base class for day counter implementations ) . . . . .	356
<a href="#">DEMCurrency</a> (Deutsche mark ) . . . . .	357
<a href="#">Denmark</a> (Danish calendar ) . . . . .	358
<a href="#">DepositRateHelper</a> (Deposit rate helper ) . . . . .	359
<a href="#">DerivedQuote</a> (Market element whose value depends on another market element ) . . .	361
<a href="#">DirichletBC</a> (Neumann boundary condition (i.e., constant value) ) . . . . .	362
<a href="#">Discount</a> (Discount-curve traits ) . . . . .	364
<a href="#">DiscrepancyStatistics</a> (Statistic tool for sequences with discrepancy calculation ) . . . .	365
<a href="#">DiscreteAveragingAsianOption</a> (Discrete-averaging Asian option ) . . . . .	366
<a href="#">DiscreteAveragingAsianOption::arguments</a> (Extra arguments for single-asset discrete-average Asian option ) . . . . .	368
<a href="#">DiscreteAveragingAsianOption::engine</a> (Discrete-averaging Asian engine base class ) . .	369
<a href="#">DiscreteGeometricASO</a> (Discrete geometric average-strike Asian option (European style) )	370
<a href="#">DiscretizedAsset</a> (Discretized asset class used by numerical methods ) . . . . .	371
<a href="#">DiscretizedDiscountBond</a> (Useful discretized discount bond asset ) . . . . .	374
<a href="#">DiscretizedOption</a> (Discretized option on a given asset ) . . . . .	375
<a href="#">Disposable</a> (Generic disposable object with move semantics ) . . . . .	377
<a href="#">Dividend</a> (Predetermined cash flow ) . . . . .	378
<a href="#">DividendVanillaOption</a> (Single-asset vanilla option (no barriers) with discrete dividends )	380
<a href="#">DividendVanillaOption::arguments</a> (Arguments for dividend vanilla option calculation )	382
<a href="#">DividendVanillaOption::engine</a> ( <a href="#">Dividend</a> vanilla option engine base class ) . . . . .	383
<a href="#">DKKCurrency</a> (Danish krone ) . . . . .	384
<a href="#">DKKLibor</a> (DKK LIBOR rate ) . . . . .	385
<a href="#">DMinus</a> ( $D_-$ matricial representation ) . . . . .	386
<a href="#">DownRounding</a> (Down-rounding ) . . . . .	387
<a href="#">DPlus</a> ( $D_+$ matricial representation ) . . . . .	388
<a href="#">DPlusDMinus</a> ( $D_+D_-$ matricial representation ) . . . . .	389
<a href="#">DriftTermStructure</a> (Drift term structure ) . . . . .	390



<a href="#">Duration</a> ( <a href="#">Duration</a> type ) . . . . .	392
<a href="#">DZero</a> ( $D_0$ matricial representation ) . . . . .	393
<a href="#">EarlyExercise</a> (Early-exercise base class ) . . . . .	394
<a href="#">EEKCurrency</a> (Estonian kroon ) . . . . .	395
<a href="#">EndCriteria</a> (Criteria to end optimization process ) . . . . .	396
<a href="#">EqualJumpsBinomialTree</a> (Base class for equal jumps binomial tree ) . . . . .	398
<a href="#">EqualProbabilitiesBinomialTree</a> (Base class for equal probabilities binomial tree ) . . . . .	399
<a href="#">Error</a> (Base error class ) . . . . .	400
<a href="#">ErrorFunction</a> (Error function ) . . . . .	401
<a href="#">ESPCurrency</a> (Spanish peseta ) . . . . .	402
<a href="#">EulerDiscretization</a> (Euler discretization for stochastic processes ) . . . . .	403
<a href="#">EURCurrency</a> (European Euro ) . . . . .	405
<a href="#">Euribor</a> (Euribor index ) . . . . .	406
<a href="#">EURLibor</a> (EUR LIBOR rate ) . . . . .	407
<a href="#">EuropeanExercise</a> (European exercise ) . . . . .	408
<a href="#">EuropeanOption</a> (European option on a single asset ) . . . . .	409
<a href="#">Event</a> (Base class for event ) . . . . .	410
<a href="#">ExchangeRate</a> (Exchange rate between two currencies ) . . . . .	412
<a href="#">ExchangeRateManager</a> (Exchange-rate repository ) . . . . .	414
<a href="#">Exercise</a> (Base exercise class ) . . . . .	416
<a href="#">ExplicitEuler</a> (Forward Euler scheme for finite difference methods ) . . . . .	417
<a href="#">ExtendedCoxIngersollRoss</a> (Extended Cox-Ingersoll-Ross model class ) . . . . .	419
<a href="#">ExtendedCoxIngersollRoss::Dynamics</a> (Short-rate dynamics in the extended Cox-Ingersoll-Ross model ) . . . . .	421
<a href="#">ExtendedCoxIngersollRoss::FittingParameter</a> (Analytical term-structure fitting parameter $\varphi(t)$ ) . . . . .	422
<a href="#">ExtendedDiscountCurve</a> (Term structure based on loglinear interpolation of discount factors ) . . . . .	423
<a href="#">Extrapolator</a> (Base class for classes possibly allowing extrapolation ) . . . . .	425
<a href="#">Factorial</a> (Factorial numbers calculator ) . . . . .	426
<a href="#">FalsePosition</a> (False position 1-D solver ) . . . . .	427
<a href="#">FaureRsg</a> (Faure low-discrepancy sequence generator ) . . . . .	428
<a href="#">FDAmericanCondition</a> . . . . .	429
<a href="#">FDBermudanEngine</a> (Finite-differences Bermudan engine ) . . . . .	430
<a href="#">FDDividendEngineMerton73</a> (Finite-differences pricing engine for dividend options using ) . . . . .	431
<a href="#">FDDividendEngineShiftScale</a> (Finite-differences pricing engine for dividend options using ) . . . . .	432
<a href="#">FDEuropeanEngine</a> (Pricing engine for European options using finite-differences ) . . . . .	433
<a href="#">FDStepConditionEngine</a> (Finite-differences pricing engine for American-style vanilla options ) . . . . .	434
<a href="#">FIMCurrency</a> (Finnish markka ) . . . . .	435
<a href="#">FiniteDifferenceModel</a> (Generic finite difference model ) . . . . .	436
<a href="#">Finland</a> (Finnish calendar ) . . . . .	437
<a href="#">FixedCouponBond</a> (Fixed-coupon bond ) . . . . .	438
<a href="#">FixedCouponBondHelper</a> (Fixed-coupon bond helper ) . . . . .	440
<a href="#">FixedDividend</a> (Predetermined cash flow ) . . . . .	442
<a href="#">FixedRateCoupon</a> (Coupon paying a fixed interest rate ) . . . . .	444
<a href="#">FlatForward</a> (Flat interest-rate curve ) . . . . .	446
<a href="#">FloatingRateBond</a> (Floating-rate bond ) . . . . .	448
<a href="#">FloatingRateCoupon</a> (Coupon paying a variable rate ) . . . . .	450
<a href="#">Floor</a> (Concrete floor class ) . . . . .	452
<a href="#">FloorTruncation</a> ( <a href="#">Floor</a> truncation ) . . . . .	453
<a href="#">ForwardEngine</a> (Forward engine base class ) . . . . .	454

<a href="#">ForwardFlat</a> (Forward-flat interpolation factory and traits ) . . . . .	455
<a href="#">ForwardFlatInterpolation</a> (Forward-flat interpolation between discrete points ) . . . . .	456
<a href="#">ForwardOptionArguments</a> (Arguments for forward (strike-resetting) option calculation ) . . . . .	457
<a href="#">ForwardPerformanceEngine</a> (Forward performance engine ) . . . . .	458
<a href="#">ForwardRate</a> (Forward-curve traits ) . . . . .	459
<a href="#">ForwardRateStructure</a> (Forward rate term structure ) . . . . .	460
<a href="#">ForwardSpreadedTermStructure</a> (Term structure with added spread on the instantaneous forward rate ) . . . . .	462
<a href="#">ForwardVanillaOption</a> (Forward version of a vanilla option ) . . . . .	464
<a href="#">FractionalDividend</a> (Predetermined cash flow ) . . . . .	466
<a href="#">FraRateHelper</a> (Forward rate agreement helper ) . . . . .	468
<a href="#">FRFCurrency</a> (French franc ) . . . . .	470
<a href="#">FuturesRateHelper</a> (Interest-rate futures helper ) . . . . .	471
<a href="#">G2</a> (Two-additive-factor gaussian model class ) . . . . .	472
<a href="#">G2::FittingParameter</a> (Analytical term-structure fitting parameter $\varphi(t)$ ) . . . . .	474
<a href="#">G2SwaptionEngine</a> (Swaption priced by means of the Black formula ) . . . . .	475
<a href="#">GammaFunction</a> (Gamma function class ) . . . . .	476
<a href="#">GapPayoff</a> (Binary gap payoff ) . . . . .	477
<a href="#">GaussChebyshev2thIntegration</a> (Gauss-Chebyshev integration second kind ) . . . . .	478
<a href="#">GaussChebyshevIntegration</a> (Gauss-Chebyshev integration ) . . . . .	479
<a href="#">GaussGegenbauerIntegration</a> (Gauss-Gegenbauer integration ) . . . . .	480
<a href="#">GaussHermiteIntegration</a> (Generalized Gauss-Hermite integration ) . . . . .	481
<a href="#">GaussHermitePolynomial</a> (Gauss-Hermite polynomial ) . . . . .	482
<a href="#">GaussHyperbolicIntegration</a> (Gauss-Hyperbolic integration ) . . . . .	483
<a href="#">GaussHyperbolicPolynomial</a> (Gauss hyperbolic polynomial ) . . . . .	484
<a href="#">GaussianOrthogonalPolynomial</a> (Orthogonal polynomial for Gaussian quadratures ) . . . . .	485
<a href="#">GaussianQuadrature</a> (Integral of a 1-dimensional function using the Gauss quadratures method ) . . . . .	486
<a href="#">GaussianStatistics</a> (Statistics tool for gaussian-assumption risk measures ) . . . . .	487
<a href="#">GaussJacobiIntegration</a> (Gauss-Jacobi integration ) . . . . .	490
<a href="#">GaussJacobiPolynomial</a> (Gauss-Jacobi polynomial ) . . . . .	491
<a href="#">GaussLaguerreIntegration</a> (Generalized Gauss-Laguerre integration ) . . . . .	492
<a href="#">GaussLaguerrePolynomial</a> (Gauss-Laguerre polynomial ) . . . . .	493
<a href="#">GaussLegendreIntegration</a> (Gauss-Legendre integration ) . . . . .	494
<a href="#">GBPCurrency</a> (British pound sterling ) . . . . .	495
<a href="#">GBPLibor</a> (GBP LIBOR rate ) . . . . .	496
<a href="#">GeneralStatistics</a> (Statistics tool ) . . . . .	497
<a href="#">GenericEngine</a> (Template base class for option pricing engines ) . . . . .	500
<a href="#">GenericModelEngine</a> (Base class for some pricing engine on a particular model ) . . . . .	501
<a href="#">GenericRiskStatistics</a> (Empirical-distribution risk measures ) . . . . .	502
<a href="#">GeometricBrownianMotionProcess</a> (Geometric brownian-motion process ) . . . . .	505
<a href="#">Germany</a> (German calendars ) . . . . .	506
<a href="#">GRDCurrency</a> (Greek drachma ) . . . . .	509
<a href="#">Greeks</a> (Additional option results ) . . . . .	510
<a href="#">HaltonRsg</a> (Halton low-discrepancy sequence generator ) . . . . .	511
<a href="#">Handle</a> (Globally accessible relinkable pointer ) . . . . .	512
<a href="#">HestonModel</a> (Heston model for the stochastic volatility of an asset ) . . . . .	513
<a href="#">HestonModelHelper</a> (Calibration helper for Heston model ) . . . . .	514
<a href="#">HestonProcess</a> (Square-root stochastic-volatility Heston process ) . . . . .	515
<a href="#">History</a> (Container for historical data ) . . . . .	517
<a href="#">History::const_iterator</a> (Random access iterator on history entries ) . . . . .	520
<a href="#">History::Entry</a> (Single datum in history ) . . . . .	521
<a href="#">HKDCurrency</a> (Honk Kong dollar ) . . . . .	522
<a href="#">HongKong</a> (Hong Kong calendars ) . . . . .	523

HUFCurrency (Hungarian forint)	525
HullWhite (Single-factor Hull-White (extended Vasicek) model class)	526
HullWhite::Dynamics (Short-rate dynamics in the Hull-White model)	528
HullWhite::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$ )	529
Hungary (Hungarian calendar)	530
Iceland (Icelandic calendars)	531
IEPCurrency (Irish punt)	533
ILSCurrency (Israeli shekel)	534
IMM (Main cycle of the International Money Market (a.k.a. IMM) Months)	535
ImplicitEuler (Backward Euler scheme for finite difference methods)	536
ImpliedTermStructure (Implied term structure at a given date in the future)	537
ImpliedVolTermStructure (Implied vol term structure at a given date in the future)	539
InArrearIndexedCoupon (In-arrear floating-rate coupon)	541
IncrementalStatistics (Statistics tool based on incremental accumulation)	543
Index (Purely virtual base class for indexes)	546
IndexedCoupon (Base indexed coupon class)	547
IndexManager (Global repository for past index fixings)	549
India (Indian calendars)	550
Indonesia (Indonesian calendars)	552
INRCurrency (Indian rupee)	554
Instrument (Abstract instrument class)	555
IntegralEngine	558
InterestRate (Concrete interest rate class)	559
InterpolatedDiscountCurve (Term structure based on interpolation of discount factors)	562
InterpolatedForwardCurve (Term structure based on interpolation of forward rates)	564
InterpolatedZeroCurve (Term structure based on interpolation of zero yields)	566
Interpolation (Base class for 1-D interpolations)	568
Interpolation2D (Base class for 2-D interpolations)	569
Interpolation2D::templateImpl (Basic template implementation)	570
Interpolation2DImpl (Abstract base class for 2-D interpolation implementations)	571
Interpolation::templateImpl (Basic template implementation)	572
InterpolationImpl (Abstract base class for interpolation implementations)	573
InverseCumulativeNormal (Inverse cumulative normal distribution function)	574
InverseCumulativePoisson (Inverse cumulative Poisson distribution function)	575
InverseCumulativeRng (Inverse cumulative random number generator)	576
InverseCumulativeRsg (Inverse cumulative random sequence generator)	577
IQDCurrency (Iraqi dinar)	578
IRRCurrency (Iranian rial)	579
ISKCurrency (Iceland krona)	580
Italy (Italian calendars)	581
ITLCurrency (Italian lira)	583
JamshidianSwaptionEngine (Jamshidian swaption engine)	584
Japan (Japanese calendar)	585
JarrowRudd (Jarrow-Rudd (multiplicative) equal probabilities binomial tree)	587
Jibar (JIBAR rate)	588
JointCalendar (Joint calendar)	589
JPYCurrency (Japanese yen)	590
JPYLibor (JPY LIBOR rate)	591
JumpDiffusionEngine (Jump-diffusion engine for vanilla options)	592
JuQuadraticApproximationEngine	593
KnuthUniformRng (Uniform random number generator)	594
KronrodIntegral (Integral of a 1-dimensional function using the Gauss-Kronrod method)	595
KRWCurrency (South-Korean won)	596
KWDCurrency (Kuwaiti dinar)	597

<a href="#">Lattice</a> (Lattice-method base class ) . . . . .	598
<a href="#">Lattice1D</a> (One-dimensional lattice ) . . . . .	600
<a href="#">Lattice2D</a> (Two-dimensional lattice ) . . . . .	601
<a href="#">LatticeShortRateModelEngine</a> (Engine for a short-rate model specialized on a lattice ) . . . . .	602
<a href="#">LazyObject</a> (Framework for calculation on demand and result caching ) . . . . .	603
<a href="#">LeastSquareFunction</a> (Cost function for least-square problems ) . . . . .	605
<a href="#">LeastSquareProblem</a> (Base class for least square problem ) . . . . .	606
<a href="#">LecuyerUniformRng</a> (Uniform random number generator ) . . . . .	607
<a href="#">LeisenReimer</a> (Leisen & Reimer tree: multiplicative approach ) . . . . .	608
<a href="#">LevenbergMarquardt</a> (Levenberg-Marquardt optimization method ) . . . . .	609
<a href="#">LexicographicalView</a> (Lexicographical 2-D view of a contiguous set of data ) . . . . .	610
<a href="#">LfmCovarianceParameterization</a> ( <a href="#">Libor</a> market model parameterization ) . . . . .	612
<a href="#">LfmCovarianceProxy</a> (Proxy for a libor forward model covariance parameterization ) . . . . .	613
<a href="#">LfmHullWhiteParameterization</a> ( <a href="#">Libor</a> market model parameterization based on Hull White paper ) . . . . .	614
<a href="#">LfmSwaptionEngine</a> ( <a href="#">Libor</a> forward model swaption engine based on black formula ) . . . . .	615
<a href="#">Libor</a> (Base class for BBA LIBOR indexes ) . . . . .	616
<a href="#">LiborForwardModel</a> ( <a href="#">Libor</a> Forward Model ) . . . . .	617
<a href="#">LiborForwardModelProcess</a> (Libor-forward-model process ) . . . . .	619
<a href="#">Linear</a> ( <a href="#">Linear</a> interpolation factory and traits ) . . . . .	621
<a href="#">LinearInterpolation</a> (Linear interpolation between discrete points ) . . . . .	622
<a href="#">LineSearch</a> (Base class for line search ) . . . . .	623
<a href="#">Link</a> (Relinkable access to a shared pointer ) . . . . .	625
<a href="#">LmCorrelationModel</a> ( <a href="#">Libor</a> forward correlation model ) . . . . .	627
<a href="#">LmExponentialCorrelationModel</a> (Exponential correlation model ) . . . . .	628
<a href="#">LmLinearExponentialVolatilityModel</a> ( <a href="#">Linear</a> exponential volatility model ) . . . . .	629
<a href="#">LmVolatilityModel</a> (Caplet volatility model ) . . . . .	630
<a href="#">LocalConstantVol</a> (Constant local volatility, no time-strike dependence ) . . . . .	631
<a href="#">LocalVolCurve</a> (Local volatility curve derived from a Black curve ) . . . . .	632
<a href="#">LocalVolSurface</a> (Local volatility surface derived from a Black vol surface ) . . . . .	634
<a href="#">LocalVolTermStructure</a> (Local-volatility term structure ) . . . . .	636
<a href="#">LogLinear</a> (Log-linear interpolation factory and traits ) . . . . .	638
<a href="#">LogLinearInterpolation</a> . . . . .	639
<a href="#">LTLCurrency</a> (Lithuanian litas ) . . . . .	640
<a href="#">LUFCurrency</a> (Luxembourg franc ) . . . . .	641
<a href="#">LVLCurrency</a> (Latvian lat ) . . . . .	642
<a href="#">MakeMCDigitalEngine</a> (Monte Carlo digital engine factory ) . . . . .	643
<a href="#">MakeMCEuropeanEngine</a> (Monte Carlo European engine factory ) . . . . .	644
<a href="#">MakeMCEuropeanHestonEngine</a> (Monte Carlo Heston European engine factory ) . . . . .	645
<a href="#">MakeSchedule</a> (Helper class ) . . . . .	646
<a href="#">Matrix</a> (Matrix used in linear algebra ) . . . . .	647
<a href="#">MCAmericanBasketEngine</a> (Least-square Monte Carlo engine ) . . . . .	651
<a href="#">MCBarrierEngine</a> (Pricing engine for barrier options using Monte Carlo simulation ) . . . . .	652
<a href="#">MCBasketEngine</a> (Pricing engine for basket options using Monte Carlo simulation ) . . . . .	654
<a href="#">McCliquetOption</a> (Simple example of Monte Carlo pricer ) . . . . .	656
<a href="#">MCDigitalEngine</a> (Pricing engine for digital options using Monte Carlo simulation ) . . . . .	657
<a href="#">MCDiscreteArithmeticAPEngine</a> (Monte Carlo pricing engine for discrete arithmetic average price Asian ) . . . . .	658
<a href="#">McDiscreteArithmeticASO</a> (Example of Monte Carlo pricer using a control variate ) . . . . .	660
<a href="#">MCDiscreteAveragingAsianEngine</a> (Pricing engine for discrete average Asians using Monte Carlo simulation ) . . . . .	661
<a href="#">MCDiscreteGeometricAPEngine</a> (Monte Carlo pricing engine for discrete geometric av- erage price Asian ) . . . . .	663
<a href="#">MCEuropeanEngine</a> (European option pricing engine using Monte Carlo simulation ) . . . . .	664

<a href="#">MCEuropeanHestonEngine</a> (Monte Carlo Heston-model engine for European options )	665
<a href="#">McEverest</a> (Everest-type option pricer )	666
<a href="#">McHimalaya</a> (Himalayan-type option pricer )	667
<a href="#">McMaxBasket</a> (Simple example of multi-factor Monte Carlo pricer )	668
<a href="#">McPagoda</a> (Roofed Asian option )	669
<a href="#">McPerformanceOption</a> (Performance option computed using Monte Carlo simulation )	670
<a href="#">McPricer</a> (Base class for Monte Carlo pricers )	671
<a href="#">McSimulation</a> (Base class for Monte Carlo engines )	672
<a href="#">MCVanillaEngine</a> (Pricing engine for vanilla options using Monte Carlo simulation )	674
<a href="#">MersenneTwisterUniformRng</a> (Uniform random number generator )	675
<a href="#">Merton76Process</a> (Merton-76 jump-diffusion process )	676
<a href="#">Mexico</a> (Mexican calendars )	678
<a href="#">MixedScheme</a> (Mixed (explicit/implicit) scheme for finite difference methods )	680
<a href="#">Money</a> (Amount of cash )	682
<a href="#">MonotonicCubicSpline</a> (Cubic spline with monotonicity constraint )	684
<a href="#">MonteCarloModel</a> (General purpose Monte Carlo model for path samples )	685
<a href="#">MoreGreeks</a> (More additional option results )	686
<a href="#">MoroInverseCumulativeNormal</a> (Moro Inverse cumulative normal distribution class )	687
<a href="#">MTLCurrency</a> (Maltese lira )	688
<a href="#">MultiAssetOption</a> (Base class for options on multiple assets )	689
<a href="#">MultiAssetOption::arguments</a> (Arguments for multi-asset option calculation )	691
<a href="#">MultiAssetOption::results</a> (Results from multi-asset option calculation )	692
<a href="#">MultiCubicSpline</a>	693
<a href="#">MultiPath</a> (Correlated multiple asset paths )	694
<a href="#">MultiPathGenerator</a> (Generates a multipath from a random number generator )	695
<a href="#">MultiVariate</a> (Default Monte Carlo traits for multi-variate models )	696
<a href="#">MXNCurrency</a> (Mexican peso )	697
<a href="#">NaturalCubicSpline</a> (Cubic spline with null second derivative at end points )	698
<a href="#">NaturalMonotonicCubicSpline</a> (Natural cubic spline with monotonicity constraint )	699
<a href="#">NeumannBC</a> (Neumann boundary condition (i.e., constant derivative) )	700
<a href="#">Newton</a> (Newton 1-D solver )	702
<a href="#">NewtonSafe</a> (Safe Newton 1-D solver )	703
<a href="#">NewZealand</a> (New Zealand calendar )	704
<a href="#">NLGCurrency</a> (Dutch guilder )	705
<a href="#">NoConstraint</a> (No constraint )	706
<a href="#">NOKCurrency</a> (Norwegian krone )	707
<a href="#">NonLinearLeastSquare</a> (Non-linear least-square method )	708
<a href="#">NormalDistribution</a> (Normal distribution function )	709
<a href="#">Norway</a> (Norwegian calendar )	710
<a href="#">NPRCurrency</a> (Nepal rupee )	711
<a href="#">Null</a> (Template class providing a null value for a given type )	712
<a href="#">NullCalendar</a> (Calendar for reproducing theoretical calculations )	713
<a href="#">NullCondition</a> (Null step condition )	714
<a href="#">NullParameter</a> (Parameter which is always zero $a(t) = 0$ )	715
<a href="#">NumericalMethod</a> (Numerical method (tree, finite-differences) base class )	716
<a href="#">NZDCurrency</a> (New Zealand dollar )	718
<a href="#">NZDLibor</a> (NZD LIBOR rate )	719
<a href="#">Observable</a> (Object that notifies its changes to a set of observables )	720
<a href="#">ObservableValue</a> ( <a href="#">Observable</a> and assignable proxy to concrete value )	721
<a href="#">Observer</a> (Object that gets notified when a given observable changes )	722
<a href="#">OneAssetOption</a> (Base class for options on a single asset )	724
<a href="#">OneAssetOption::arguments</a> (Arguments for single-asset option calculation )	727
<a href="#">OneAssetOption::results</a> (Results from single-asset option calculation )	728
<a href="#">OneAssetStrikedOption</a> (Base class for options on a single asset with striked payoff )	729



<a href="#">OneDayCounter</a> (1/1 day count convention ) . . . . .	731
<a href="#">OneFactorAffineModel</a> (Single-factor affine base class ) . . . . .	732
<a href="#">OneFactorModel</a> (Single-factor short-rate model abstract class ) . . . . .	733
<a href="#">OneFactorModel::ShortRateDynamics</a> (Base class describing the short-rate dynamics ) . . . . .	734
<a href="#">OneFactorModel::ShortRateTree</a> (Recombining trinomial tree discretizing the state variable ) . . . . .	735
<a href="#">OperatorFactory</a> (Black-Scholes-Merton differential operator ) . . . . .	736
<a href="#">OptimizationMethod</a> (Abstract class for constrained optimization method ) . . . . .	737
<a href="#">Option</a> (Base option class ) . . . . .	739
<a href="#">Option::arguments</a> . . . . .	740
<a href="#">OrnsteinUhlenbeckProcess</a> (Ornstein-Uhlenbeck process class ) . . . . .	741
<a href="#">Parameter</a> (Base class for model arguments ) . . . . .	743
<a href="#">ParameterImpl</a> (Base class for model parameter implementation ) . . . . .	744
<a href="#">ParCoupon</a> (coupon at par on a term structure ) . . . . .	745
<a href="#">Path</a> . . . . .	747
<a href="#">PathGenerator</a> (Generates random paths using a sequence generator ) . . . . .	748
<a href="#">PathPricer</a> (Base class for path pricers ) . . . . .	749
<a href="#">Payoff</a> (Base class for option payoffs ) . . . . .	750
<a href="#">PercentageStrikePayoff</a> (Payoff with strike expressed as percentage ) . . . . .	751
<a href="#">Period</a> (Time period described by a number of a given time unit ) . . . . .	752
<a href="#">PiecewiseConstantParameter</a> (Piecewise-constant parameter ) . . . . .	753
<a href="#">PiecewiseYieldCurve</a> (Piecewise yield term structure ) . . . . .	754
<a href="#">PKRCurrency</a> (Pakistani rupee ) . . . . .	756
<a href="#">PlainVanillaPayoff</a> (Plain-vanilla payoff ) . . . . .	757
<a href="#">PLNCurrency</a> (Polish zloty ) . . . . .	758
<a href="#">PoissonDistribution</a> (Normal distribution function ) . . . . .	759
<a href="#">Poland</a> (Polish calendar ) . . . . .	760
<a href="#">PositiveConstraint</a> (Constraint imposing positivity to all arguments ) . . . . .	761
<a href="#">PriceCurve</a> (Additional pricing results ) . . . . .	762
<a href="#">PricingEngine</a> (Interface for pricing engines ) . . . . .	763
<a href="#">PrimeNumbers</a> (Prime numbers calculator ) . . . . .	764
<a href="#">Problem</a> (Constrained optimization problem ) . . . . .	765
<a href="#">PTECurrency</a> (Portuguese escudo ) . . . . .	767
<a href="#">QuantoEngine</a> (Quanto engine base class ) . . . . .	768
<a href="#">QuantoForwardVanillaOption</a> (Quanto version of a forward vanilla option ) . . . . .	770
<a href="#">QuantoOptionArguments</a> (Arguments for quanto option calculation ) . . . . .	772
<a href="#">QuantoOptionResults</a> (Results from quanto option calculation ) . . . . .	773
<a href="#">QuantoTermStructure</a> (Quanto term structure ) . . . . .	774
<a href="#">QuantoVanillaOption</a> (Quanto version of a vanilla option ) . . . . .	776
<a href="#">Quote</a> (Purely virtual base class for market observables ) . . . . .	778
<a href="#">RandomizedLDS</a> (Randomized (random shift) low-discrepancy sequence ) . . . . .	779
<a href="#">RandomSequenceGenerator</a> (Random sequence generator based on a pseudo-random number generator ) . . . . .	781
<a href="#">RateHelper</a> (Base class for rate helpers ) . . . . .	782
<a href="#">Results</a> (Base class for generic result groups ) . . . . .	784
<a href="#">Ridder</a> (Ridder 1-D solver ) . . . . .	785
<a href="#">ROLCurrency</a> (Romanian leu ) . . . . .	786
<a href="#">Rounding</a> (Basic rounding class ) . . . . .	787
<a href="#">SalvagingAlgorithm</a> (Algorithm used for matricial pseudo square root ) . . . . .	789
<a href="#">Sample</a> (Weighted sample ) . . . . .	790
<a href="#">SampledCurve</a> (This class contains a sampled curve ) . . . . .	791
<a href="#">SARCurrency</a> (Saudi riyal ) . . . . .	793
<a href="#">SaudiArabia</a> (Saudi Arabian calendar ) . . . . .	794
<a href="#">Schedule</a> (Payment schedule ) . . . . .	795

<a href="#">Secant</a> (Secant 1-D solver ) . . . . .	796
<a href="#">SeedGenerator</a> (Random seed generator ) . . . . .	797
<a href="#">SegmentIntegral</a> (Integral of a one-dimensional function ) . . . . .	798
<a href="#">SEKCurrency</a> (Swedish krona ) . . . . .	799
<a href="#">SequenceStatistics</a> (Statistics analysis of N-dimensional (sequence) data ) . . . . .	800
<a href="#">Settings</a> (Global repository for run-time library settings ) . . . . .	802
<a href="#">SGDCurrency</a> ( <a href="#">Singapore</a> dollar ) . . . . .	804
<a href="#">Short</a> (Short indexed coupon ) . . . . .	805
<a href="#">Short&lt; ParCoupon &gt;</a> (Short coupon at par on a term structure ) . . . . .	806
<a href="#">ShortRateModel</a> (Abstract short-rate model class ) . . . . .	807
<a href="#">ShoutCondition</a> (Shout option condition ) . . . . .	809
<a href="#">SimpleCashFlow</a> (Predetermined cash flow ) . . . . .	810
<a href="#">SimpleDayCounter</a> (Simple day counter for reproducing theoretical calculations ) . . . . .	812
<a href="#">SimpleQuote</a> (Market element returning a stored value ) . . . . .	813
<a href="#">Simplex</a> (Multi-dimensional simplex class ) . . . . .	814
<a href="#">SimpsonIntegral</a> (Integral of a one-dimensional function ) . . . . .	815
<a href="#">Singapore</a> (Singapore calendars ) . . . . .	816
<a href="#">SingleAssetOption</a> (Black-Scholes-Merton option ) . . . . .	818
<a href="#">Singleton</a> (Basic support for the singleton pattern ) . . . . .	820
<a href="#">SingleVariate</a> (Default Monte Carlo traits for single-variate models ) . . . . .	821
<a href="#">SITCurrency</a> (Slovenian tolar ) . . . . .	822
<a href="#">SKKCurrency</a> (Slovak koruna ) . . . . .	823
<a href="#">Slovakia</a> (Slovak calendars ) . . . . .	824
<a href="#">SobolRsg</a> (Sobol low-discrepancy sequence generator ) . . . . .	826
<a href="#">Solver1D</a> (Base class for 1-D solvers ) . . . . .	828
<a href="#">SouthAfrica</a> (South-African calendar ) . . . . .	830
<a href="#">SouthKorea</a> (South Korean calendars ) . . . . .	831
<a href="#">SquareRootProcess</a> (Square-root process class ) . . . . .	833
<a href="#">StatsHolder</a> (Helper class for precomputed distributions ) . . . . .	834
<a href="#">SteepestDescent</a> (Multi-dimensional steepest-descent class ) . . . . .	835
<a href="#">step_iterator</a> (Iterator advancing in constant steps ) . . . . .	836
<a href="#">StepCondition</a> (Condition to be applied at every time step ) . . . . .	837
<a href="#">StepConditionSet</a> (Parallel evolver for multiple arrays ) . . . . .	838
<a href="#">StochasticProcess</a> (Multi-dimensional stochastic process class ) . . . . .	839
<a href="#">StochasticProcess1D</a> (1-dimensional stochastic process ) . . . . .	842
<a href="#">StochasticProcess1D::discretization</a> (Discretization of a 1-D stochastic process ) . . . . .	844
<a href="#">StochasticProcess::discretization</a> (Discretization of a stochastic process over a given time interval ) . . . . .	845
<a href="#">StochasticProcessArray</a> ( <a href="#">Array</a> of correlated 1-D stochastic processes ) . . . . .	846
<a href="#">Stock</a> (Simple stock class ) . . . . .	848
<a href="#">StrikedTypePayoff</a> (Intermediate class for payoffs based on a fixed strike ) . . . . .	849
<a href="#">StulzEngine</a> (Pricing engine for 2D European Baskets ) . . . . .	850
<a href="#">SuperSharePayoff</a> (Binary supershare payoff ) . . . . .	851
<a href="#">SVD</a> (Singular value decomposition ) . . . . .	852
<a href="#">Swap</a> (Interest rate swap ) . . . . .	853
<a href="#">SwapRateHelper</a> (Swap rate helper ) . . . . .	855
<a href="#">Swaption</a> (Swaption class ) . . . . .	858
<a href="#">Swaption::arguments</a> (Arguments for swaption calculation ) . . . . .	860
<a href="#">Swaption::results</a> (Results from swaption calculation ) . . . . .	861
<a href="#">SwaptionHelper</a> (Calibration helper for ATM swaption ) . . . . .	862
<a href="#">SwaptionVolatilityMatrix</a> (At-the-money swaption-volatility matrix ) . . . . .	863
<a href="#">SwaptionVolatilityStructure</a> (Swaption-volatility structure ) . . . . .	865
<a href="#">Sweden</a> (Swedish calendar ) . . . . .	867
<a href="#">Switzerland</a> (Swiss calendar ) . . . . .	868

<a href="#">SymmetricSchurDecomposition</a> (Symmetric threshold Jacobi algorithm ) . . . . .	869
<a href="#">TabulatedGaussLegendre</a> (Tabulated Gauss-Legendre quadratures ) . . . . .	870
<a href="#">Taiwan</a> (Taiwanese calendars ) . . . . .	871
<a href="#">TARGET</a> (TARGET calendar ) . . . . .	873
<a href="#">TermStructure</a> (Basic term-structure functionality ) . . . . .	874
<a href="#">TermStructureConsistentModel</a> (Term-structure consistent model class ) . . . . .	876
<a href="#">TermStructureFittingParameter</a> (Deterministic time-dependent parameter used for yield-curve fitting ) . . . . .	877
<a href="#">THBCurrency</a> (Thai baht ) . . . . .	878
<a href="#">Thirty360</a> (30/360 day count convention ) . . . . .	879
<a href="#">Tian</a> (Tian tree: third moment matching, multiplicative approach ) . . . . .	880
<a href="#">Tibor</a> (JPY TIBOR index ) . . . . .	881
<a href="#">TimeBasket</a> (Distribution over a number of dates ) . . . . .	882
<a href="#">TimeGrid</a> (Time grid class ) . . . . .	883
<a href="#">TqrEigenDecomposition</a> (Tridiag. QR eigen decomposition with explicite shift aka Wilkinson ) . . . . .	885
<a href="#">TransformedGrid</a> (Transformed grid ) . . . . .	886
<a href="#">TrapezoidIntegral</a> (Integral of a one-dimensional function ) . . . . .	887
<a href="#">Tree</a> (Tree approximating a single-factor diffusion ) . . . . .	889
<a href="#">TreeCapFloorEngine</a> (Numerical lattice engine for cap/floors ) . . . . .	890
<a href="#">TreeSwaptionEngine</a> (Numerical lattice engine for swaptions ) . . . . .	891
<a href="#">TreeVanillaSwapEngine</a> (Numerical lattice engine for simple swaps ) . . . . .	892
<a href="#">TridiagonalOperator</a> (Base implementation for tridiagonal operator ) . . . . .	893
<a href="#">TridiagonalOperator::TimeSetter</a> (Encapsulation of time-setting logic ) . . . . .	895
<a href="#">Trigeorgis</a> (Trigeorgis (additive equal jumps) binomial tree ) . . . . .	896
<a href="#">TrinomialTree</a> (Recombining trinomial tree class ) . . . . .	897
<a href="#">TRLCurrency</a> (Turkish lira ) . . . . .	898
<a href="#">TRLibor</a> (TRY LIBOR rate ) . . . . .	899
<a href="#">TRYCurrency</a> (New Turkish lira ) . . . . .	900
<a href="#">TsiveriotisFernandesLattice</a> (Binomial lattice approximating the Tsiveriotis-Fernandes model ) . . . . .	901
<a href="#">TTDCurrency</a> (Trinidad & Tobago dollar ) . . . . .	903
<a href="#">Turkey</a> (Turkish calendar ) . . . . .	904
<a href="#">TWDCurrency</a> (Taiwan dollar ) . . . . .	905
<a href="#">TwoFactorModel</a> (Abstract base-class for two-factor models ) . . . . .	906
<a href="#">TwoFactorModel::ShortRateDynamics</a> (Class describing the dynamics of the two state variables ) . . . . .	907
<a href="#">TwoFactorModel::ShortRateTree</a> (Recombining two-dimensional tree discretizing the state variable ) . . . . .	908
<a href="#">TypePayoff</a> (Intermediate class for call/put/straddle payoffs ) . . . . .	909
<a href="#">Ukraine</a> (Ukrainian calendars ) . . . . .	910
<a href="#">UnitedKingdom</a> (United Kingdom calendars ) . . . . .	912
<a href="#">UnitedStates</a> (United States calendars ) . . . . .	914
<a href="#">UpFrontIndexedCoupon</a> (up front indexed coupon class ) . . . . .	917
<a href="#">UpRounding</a> (Up-rounding ) . . . . .	919
<a href="#">USDCurrency</a> (U.S. dollar ) . . . . .	920
<a href="#">USDLibor</a> (USD LIBOR rate ) . . . . .	921
<a href="#">Value</a> (Pricing results ) . . . . .	922
<a href="#">VanillaOption</a> (Vanilla option (no discrete dividends, no barriers) on a single asset ) . .	923
<a href="#">VanillaOption::engine</a> (Vanilla option engine base class ) . . . . .	924
<a href="#">VanillaSwap</a> (Plain-vanilla swap ) . . . . .	925
<a href="#">VanillaSwap::arguments</a> (Arguments for simple swap calculation ) . . . . .	927
<a href="#">VanillaSwap::results</a> (Results from simple swap calculation ) . . . . .	928
<a href="#">Vasicek</a> (Vasicek model class ) . . . . .	929



<a href="#">Vasicek::Dynamics</a> (Short-rate dynamics in the Vasicek model ) . . . . .	931
<a href="#">VEBCurrency</a> (Venezuelan bolivar ) . . . . .	932
<a href="#">Visitor</a> (Visitor for a specific class ) . . . . .	933
<a href="#">Xibor</a> (Base class for LIBOR-like indexes ) . . . . .	934
<a href="#">YieldTermStructure</a> (Interest-rate term structure ) . . . . .	936
<a href="#">ZARCurrency</a> (South-African rand ) . . . . .	940
<a href="#">ZeroCondition</a> (Zero exercise condition ) . . . . .	941
<a href="#">ZeroCouponBond</a> (Zero-coupon bond ) . . . . .	942
<a href="#">ZeroSpreadedTermStructure</a> (Term structure with an added spread on the zero yield rate )	943
<a href="#">ZeroYield</a> (Zero-curve traits ) . . . . .	945
<a href="#">ZeroYieldStructure</a> (Zero-yield term structure ) . . . . .	946
<a href="#">Zibor</a> (CHF ZIBOR rate ) . . . . .	948



## 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 ) . . . . .	949
<a href="#">ql/calendar.hpp</a> (calendar class ) . . . . .	951
<a href="#">ql/capvolstructures.hpp</a> (Cap/Floor volatility structures ) . . . . .	989
<a href="#">ql/cashflow.hpp</a> (Base class for cash flows ) . . . . .	990
<a href="#">ql/currency.hpp</a> (Known currencies ) . . . . .	1017
<a href="#">ql/date.hpp</a> (Date- and time-related classes, typedefs and enumerations ) . . . . .	1018
<a href="#">ql/daycounter.hpp</a> (Day counter class ) . . . . .	1021
<a href="#">ql/discretizedasset.hpp</a> (Discretized asset classes ) . . . . .	1028
<a href="#">ql/errors.hpp</a> (Classes and functions for error handling ) . . . . .	1029
<a href="#">ql/event.hpp</a> (Base class for events associated with a given date ) . . . . .	1032
<a href="#">ql/exchangerate.hpp</a> (Exchange rate between two currencies ) . . . . .	1033
<a href="#">ql/exercise.hpp</a> (Option exercise classes and payoff function ) . . . . .	1034
<a href="#">ql/grid.hpp</a> (Grid constructors ) . . . . .	1061
<a href="#">ql/handle.hpp</a> (Globally accessible relinkable pointer ) . . . . .	1062
<a href="#">ql/history.hpp</a> (History class ) . . . . .	1063
<a href="#">ql/index.hpp</a> (Purely virtual base class for indexes ) . . . . .	1064
<a href="#">ql/instrument.hpp</a> (Abstract instrument class ) . . . . .	1083
<a href="#">ql/interestrates.hpp</a> (Instrument rate class ) . . . . .	1111
<a href="#">ql/money.hpp</a> (Cash amount in a given currency ) . . . . .	1174
<a href="#">ql/numericalmethod.hpp</a> (Numerical method class ) . . . . .	1186
<a href="#">ql/option.hpp</a> (Base option class ) . . . . .	1200
<a href="#">ql/payoff.hpp</a> (Option payoff classes ) . . . . .	1208
<a href="#">ql/pricingengine.hpp</a> (Base class for pricing engines ) . . . . .	1219
<a href="#">ql/qldefines.hpp</a> (Global definitions and compiler switches ) . . . . .	1294
<a href="#">ql/quote.hpp</a> (Purely virtual base class for market observables ) . . . . .	1296
<a href="#">ql/schedule.hpp</a> (Date schedule ) . . . . .	1312
<a href="#">ql/settings.hpp</a> (Global repository for run-time library settings ) . . . . .	1313
<a href="#">ql/solver1d.hpp</a> (Abstract 1-D solver class ) . . . . .	1337
<a href="#">ql/stochasticprocess.hpp</a> (Stochastic processes ) . . . . .	1345
<a href="#">ql/swaptionvolstructure.hpp</a> (Swaption volatility structure ) . . . . .	1346
<a href="#">ql/termstructure.hpp</a> (Base class for term structures ) . . . . .	1347
<a href="#">ql/timegrid.hpp</a> (Discrete time grid ) . . . . .	1367
<a href="#">ql/types.hpp</a> (Custom types ) . . . . .	1368

ql/voltermstructure.hpp (Volatility term structures)	1390
ql/yieldtermstructure.hpp (Interest-rate term structure)	1391
ql/Calendars/argentina.hpp (Argentinian calendars)	953
ql/Calendars/beijing.hpp (Chinese calendar)	954
ql/Calendars/bombay.hpp (Indian calendars)	955
ql/Calendars/bratislava.hpp (Slovak calendars)	956
ql/Calendars/brazil.hpp (Brazilian calendar)	957
ql/Calendars/budapest.hpp (Hungarian calendar)	958
ql/Calendars/copenhagen.hpp (Danish calendar)	959
ql/Calendars/germany.hpp (German calendars)	960
ql/Calendars/helsinki.hpp (Finnish calendar)	961
ql/Calendars/hongkong.hpp (Hong Kong calendars)	962
ql/Calendars/iceland.hpp (Icelandic calendars)	963
ql/Calendars/indonesia.hpp (Indonesian calendars)	964
ql/Calendars/istanbul.hpp (Turkish calendar)	965
ql/Calendars/italy.hpp (Italian calendars)	966
ql/Calendars/johannesburg.hpp (South-African calendar)	967
ql/Calendars/jointcalendar.hpp (Joint calendar)	968
ql/Calendars/mexico.hpp (Mexican calendars)	969
ql/Calendars/nullcalendar.hpp (Calendar for reproducing theoretical calculations)	970
ql/Calendars/oslo.hpp (Norwegian calendar)	971
ql/Calendars/prague.hpp (Czech calendars)	972
ql/Calendars/riyadh.hpp (Saudi Arabian calendar)	973
ql/Calendars/seoul.hpp (South Korean calendars)	974
ql/Calendars/singapore.hpp (Singapore calendars)	975
ql/Calendars/stockholm.hpp (Swedish calendar)	976
ql/Calendars/sydney.hpp (Australian calendar)	977
ql/Calendars/taipei.hpp (Taipei calendar)	978
ql/Calendars/taiwan.hpp (Taiwanese calendars)	979
ql/Calendars/target.hpp (TARGET calendar)	980
ql/Calendars/tokyo.hpp (Japanese calendar)	981
ql/Calendars/toronto.hpp (Canadian calendar)	982
ql/Calendars/ukraine.hpp (Ukrainian calendars)	983
ql/Calendars/unitedkingdom.hpp (UK calendars)	984
ql/Calendars/unitedstates.hpp (US calendars)	985
ql/Calendars/warsaw.hpp (Polish calendar)	986
ql/Calendars/wellington.hpp (New Zealand calendar)	987
ql/Calendars/zurich.hpp (Swiss calendar)	988
ql/CashFlows/analysis.hpp (Cash-flow analysis functions)	991
ql/CashFlows/basispointsensitivity.hpp (Basis point sensitivity calculator)	992
ql/CashFlows/cashflowvectors.hpp (Cash flow vector builders)	993
ql/CashFlows/coupon.hpp (Coupon accruing over a fixed period)	994
ql/CashFlows/dividend.hpp (A stock dividend)	995
ql/CashFlows/fixedratecoupon.hpp (Coupon paying a fixed annual rate)	996
ql/CashFlows/floatingratecoupon.hpp (Coupon paying a variable rate)	997
ql/CashFlows/inarrearindexedcoupon.hpp (In-arrear floating-rate coupon)	998
ql/CashFlows/indexedcashflowvectors.hpp (Indexed cash-flow vector builders)	999
ql/CashFlows/indexedcoupon.hpp (Indexed coupon)	1000
ql/CashFlows/parcoupon.hpp (Coupon at par on a term structure)	1001
ql/CashFlows/shortfloatingcoupon.hpp (Short (or long) coupon at par on a term structure)	1002
ql/CashFlows/shortindexedcoupon.hpp (Short (or long) indexed coupon)	1003
ql/CashFlows/simplecashflow.hpp (Predetermined cash flow)	1004
ql/CashFlows/timebasket.hpp	1005
ql/CashFlows/upfrontindexedcoupon.hpp (Up front indexed coupon)	1006

ql/Currencies/ <a href="#">africa.hpp</a> (African currencies ) . . . . .	1007
ql/Currencies/ <a href="#">america.hpp</a> (American currencies ) . . . . .	1008
ql/Currencies/ <a href="#">asia.hpp</a> (Asian currencies ) . . . . .	1010
ql/Currencies/ <a href="#">europe.hpp</a> (European currencies ) . . . . .	1012
ql/Currencies/ <a href="#">exchangeratemanager.hpp</a> (Exchange-rate repository ) . . . . .	1015
ql/Currencies/ <a href="#">oceania.hpp</a> (Oceanian currencies ) . . . . .	1016
ql/DayCounters/ <a href="#">actual360.hpp</a> (Act/360 day counter ) . . . . .	1022
ql/DayCounters/ <a href="#">actual365fixed.hpp</a> (Actual/365 (Fixed) day counter ) . . . . .	1023
ql/DayCounters/ <a href="#">actualactual.hpp</a> (Act/act day counters ) . . . . .	1024
ql/DayCounters/ <a href="#">one.hpp</a> (1/1 day counter ) . . . . .	1025
ql/DayCounters/ <a href="#">simpledaycounter.hpp</a> (Simple day counter for reproducing theoretical calculations ) . . . . .	1026
ql/DayCounters/ <a href="#">thirty360.hpp</a> (30/360 day counters ) . . . . .	1027
ql/FiniteDifferences/ <a href="#">americancondition.hpp</a> (American option exercise condition ) . . . . .	1035
ql/FiniteDifferences/ <a href="#">boundarycondition.hpp</a> (Boundary conditions for differential operators ) . . . . .	1036
ql/FiniteDifferences/ <a href="#">bsmoperator.hpp</a> (Differential operator for Black-Scholes-Merton equation ) . . . . .	1037
ql/FiniteDifferences/ <a href="#">bsmtermoperator.hpp</a> (Differential operator for Black-Scholes-Merton equation ) . . . . .	1038
ql/FiniteDifferences/ <a href="#">cranknicolson.hpp</a> (Crank-Nicolson scheme for finite difference methods ) . . . . .	1039
ql/FiniteDifferences/ <a href="#">dminus.hpp</a> ( $D_-$ matricial representation ) . . . . .	1040
ql/FiniteDifferences/ <a href="#">dplus.hpp</a> ( $D_+$ matricial representation ) . . . . .	1041
ql/FiniteDifferences/ <a href="#">dplusdminus.hpp</a> ( $D_+D_-$ matricial representation ) . . . . .	1042
ql/FiniteDifferences/ <a href="#">dzero.hpp</a> ( $D_0$ matricial representation ) . . . . .	1043
ql/FiniteDifferences/ <a href="#">expliciteuler.hpp</a> (Explicit Euler scheme for finite difference methods ) . . . . .	1044
ql/FiniteDifferences/ <a href="#">fdtypedefs.hpp</a> (Default choices for template instantiations ) . . . . .	1045
ql/FiniteDifferences/ <a href="#">finitedifferencemodel.hpp</a> (Generic finite difference model ) . . . . .	1046
ql/FiniteDifferences/ <a href="#">impliciteuler.hpp</a> (Implicit Euler scheme for finite difference methods ) . . . . .	1047
ql/FiniteDifferences/ <a href="#">mixedscheme.hpp</a> (Mixed (explicit/implicit) scheme for finite difference methods ) . . . . .	1048
ql/FiniteDifferences/ <a href="#">onefactoroperator.hpp</a> (General differential operator for one-factor interest rate models ) . . . . .	1049
ql/FiniteDifferences/ <a href="#">operatorfactory.hpp</a> (Factory for finite difference operators ) . . . . .	1050
ql/FiniteDifferences/ <a href="#">operatortraits.hpp</a> (Differential operator traits ) . . . . .	1051
ql/FiniteDifferences/ <a href="#">parallelevolver.hpp</a> (Parallel evolver for multiple arrays ) . . . . .	1052
ql/FiniteDifferences/ <a href="#">pde.hpp</a> (General class for one dimensional PDE's ) . . . . .	1053
ql/FiniteDifferences/ <a href="#">pdebsm.hpp</a> (Black-Scholes-Merton PDE ) . . . . .	1054
ql/FiniteDifferences/ <a href="#">pdeshortrate.hpp</a> (Adapter to short rate ) . . . . .	1055
ql/FiniteDifferences/ <a href="#">shoutcondition.hpp</a> (Shout option exercise condition ) . . . . .	1056
ql/FiniteDifferences/ <a href="#">stepcondition.hpp</a> (Conditions to be applied at every time step ) . . . . .	1057
ql/FiniteDifferences/ <a href="#">tridiagonaloperator.hpp</a> (Tridiagonal operator ) . . . . .	1058
ql/FiniteDifferences/ <a href="#">valueatcenter.hpp</a> (Compute value, first, and second derivatives at grid center ) . . . . .	1059
ql/FiniteDifferences/ <a href="#">zerocondition.hpp</a> (Zero option exercise condition ) . . . . .	1060
ql/Indexes/ <a href="#">audlibor.hpp</a> (AUD LIBOR rate ) . . . . .	1065
ql/Indexes/ <a href="#">cadlibor.hpp</a> (CAD LIBOR rate ) . . . . .	1066
ql/Indexes/ <a href="#">cdor.hpp</a> (CDOR rate ) . . . . .	1067
ql/Indexes/ <a href="#">chflibor.hpp</a> (CHF LIBOR rate ) . . . . .	1068
ql/Indexes/ <a href="#">dkklibor.hpp</a> (DKK LIBOR rate ) . . . . .	1069
ql/Indexes/ <a href="#">euribor.hpp</a> (Euribor index ) . . . . .	1070

ql/Indexes/ <a href="#">eurlibor.hpp</a> (EUR LIBOR rate ) . . . . .	1071
ql/Indexes/ <a href="#">gbplibor.hpp</a> (GBP LIBOR rate ) . . . . .	1072
ql/Indexes/ <a href="#">indexmanager.hpp</a> (Global repository for past index fixings ) . . . . .	1073
ql/Indexes/ <a href="#">jibar.hpp</a> (JIBAR rate ) . . . . .	1074
ql/Indexes/ <a href="#">jpylibor.hpp</a> (JPY LIBOR rate ) . . . . .	1075
ql/Indexes/ <a href="#">libor.hpp</a> (Base class for BBA LIBOR indexes ) . . . . .	1076
ql/Indexes/ <a href="#">nzdlbor.hpp</a> (NZD LIBOR rate ) . . . . .	1077
ql/Indexes/ <a href="#">tibor.hpp</a> (JPY TIBOR rate ) . . . . .	1078
ql/Indexes/ <a href="#">trlibor.hpp</a> (TRY LIBOR rate ) . . . . .	1079
ql/Indexes/ <a href="#">usdlbor.hpp</a> (USD LIBOR rate ) . . . . .	1080
ql/Indexes/ <a href="#">xibor.hpp</a> (Base class for LIBOR-like indexes ) . . . . .	1081
ql/Indexes/ <a href="#">zibor.hpp</a> (CHF ZIBOR rate ) . . . . .	1082
ql/Instruments/ <a href="#">asianoption.hpp</a> (Asian option on a single asset ) . . . . .	1084
ql/Instruments/ <a href="#">barrieroption.hpp</a> (Barrier option on a single asset ) . . . . .	1085
ql/Instruments/ <a href="#">basketoption.hpp</a> (Basket option on a number of assets ) . . . . .	1086
ql/Instruments/ <a href="#">bond.hpp</a> (Concrete bond class ) . . . . .	1087
ql/Instruments/ <a href="#">callabilityschedule.hpp</a> (Schedule of put/call dates ) . . . . .	1088
ql/Instruments/ <a href="#">capfloor.hpp</a> (Cap and Floor class ) . . . . .	1089
ql/Instruments/ <a href="#">cliquetoption.hpp</a> (Cliquet option ) . . . . .	1090
ql/Instruments/ <a href="#">convertiblebond.hpp</a> (Convertible bond class ) . . . . .	1091
ql/Instruments/ <a href="#">dividendschedule.hpp</a> (Schedule of dividend dates ) . . . . .	1093
ql/Instruments/ <a href="#">dividendvanillaoption.hpp</a> (Vanilla option on a single asset with discrete dividends ) . . . . .	1094
ql/Instruments/ <a href="#">europeanoption.hpp</a> (European option on a single asset ) . . . . .	1095
ql/Instruments/ <a href="#">fixedcouponbond.hpp</a> (Fixed-coupon bond ) . . . . .	1096
ql/Instruments/ <a href="#">floatingratebond.hpp</a> (Floating-rate bond ) . . . . .	1097
ql/Instruments/ <a href="#">forwardvanillaoption.hpp</a> (Forward version of a vanilla option ) . . . . .	1098
ql/Instruments/ <a href="#">multiassetoption.hpp</a> (Option on multiple assets ) . . . . .	1099
ql/Instruments/ <a href="#">oneassetoption.hpp</a> (Option on a single asset ) . . . . .	1100
ql/Instruments/ <a href="#">oneassetstrikedoption.hpp</a> (Option on a single asset with striked payoff ) . . . . .	1101
ql/Instruments/ <a href="#">payoffs.hpp</a> (Payoffs for various options ) . . . . .	1102
ql/Instruments/ <a href="#">quantoforwardvanillaoption.hpp</a> (Quanto version of a forward vanilla option ) . . . . .	1103
ql/Instruments/ <a href="#">quantovanillaoption.hpp</a> (Quanto version of a vanilla option ) . . . . .	1104
ql/Instruments/ <a href="#">simpleswap.hpp</a> (Simple fixed-rate vs Libor swap ) . . . . .	1105
ql/Instruments/ <a href="#">stock.hpp</a> (Concrete stock class ) . . . . .	1106
ql/Instruments/ <a href="#">swap.hpp</a> (Interest rate swap ) . . . . .	1107
ql/Instruments/ <a href="#">swaption.hpp</a> (Swaption class ) . . . . .	1108
ql/Instruments/ <a href="#">vanillaoption.hpp</a> (Vanilla option on a single asset ) . . . . .	1109
ql/Instruments/ <a href="#">zerocouponbond.hpp</a> (Zero-coupon bond ) . . . . .	1110
ql/Lattices/ <a href="#">binomialtree.hpp</a> (Binomial tree class ) . . . . .	1112
ql/Lattices/ <a href="#">bsmlattice.hpp</a> (Binomial trees under the BSM model ) . . . . .	1114
ql/Lattices/ <a href="#">lattice.hpp</a> (Lattice method class ) . . . . .	1115
ql/Lattices/ <a href="#">lattice1d.hpp</a> (One-dimensional lattice class ) . . . . .	1116
ql/Lattices/ <a href="#">lattice2d.hpp</a> (Two-dimensional lattice class ) . . . . .	1117
ql/Lattices/ <a href="#">tflattice.hpp</a> (Binomial Tsiveriotis-Fernandes tree model ) . . . . .	1118
ql/Lattices/ <a href="#">tree.hpp</a> (Tree class ) . . . . .	1119
ql/Lattices/ <a href="#">trinomialtree.hpp</a> (Trinomial tree class ) . . . . .	1120
ql/Math/ <a href="#">array.hpp</a> (1-D array used in linear algebra ) . . . . .	1121
ql/Math/ <a href="#">backwardflatinterpolation.hpp</a> (Backward-flat interpolation between discrete points ) . . . . .	1122
ql/Math/ <a href="#">beta.hpp</a> (Beta and beta incomplete functions ) . . . . .	1123
ql/Math/ <a href="#">bicubicsplineinterpolation.hpp</a> (Bicubic spline interpolation between discrete points ) . . . . .	1124

ql/Math/ <a href="#">bilinearinterpolation.hpp</a> (Bilinear interpolation between discrete points ) . . .	1125
ql/Math/ <a href="#">binomialdistribution.hpp</a> (Binomial distribution ) . . . . .	1126
ql/Math/ <a href="#">bivariatenormaldistribution.hpp</a> (Bivariate cumulative normal distribution ) . .	1127
ql/Math/ <a href="#">chisquaredistribution.hpp</a> (Chi-square (central and non-central) distributions )	1128
ql/Math/ <a href="#">choleskydecomposition.hpp</a> (Cholesky decomposition ) . . . . .	1129
ql/Math/ <a href="#">comparison.hpp</a> (Floating-point comparisons ) . . . . .	1130
ql/Math/ <a href="#">convergencestatistics.hpp</a> (Statistics tool with risk measures ) . . . . .	1131
ql/Math/ <a href="#">cubicspline.hpp</a> (Cubic spline interpolation between discrete points ) . . . . .	1132
ql/Math/ <a href="#">discrepancystatistics.hpp</a> (Statistic tool for sequences with discrepancy calculation ) . . . . .	1133
ql/Math/ <a href="#">errorfunction.hpp</a> (Error function ) . . . . .	1134
ql/Math/ <a href="#">extrapolation.hpp</a> (Class-wide extrapolation settings ) . . . . .	1135
ql/Math/ <a href="#">factorial.hpp</a> (Factorial numbers calculator ) . . . . .	1136
ql/Math/ <a href="#">forwardflatinterpolation.hpp</a> (Forward-flat interpolation between discrete points ) . . . . .	1137
ql/Math/ <a href="#">functional.hpp</a> (Functionals and combinators not included in the STL ) . . . . .	1138
ql/Math/ <a href="#">gammadistribution.hpp</a> (Gamma distribution ) . . . . .	1139
ql/Math/ <a href="#">gaussianorthogonalpolynomial.hpp</a> (Orthogonal polynomials for gaussian quadratures ) . . . . .	1140
ql/Math/ <a href="#">gaussianquadratures.hpp</a> (Integral of a 1-dimensional function using the Gauss quadratures ) . . . . .	1141
ql/Math/ <a href="#">gaussianstatistics.hpp</a> (Statistics tool for gaussian-assumption risk measures ) .	1143
ql/Math/ <a href="#">generalstatistics.hpp</a> (Statistics tool ) . . . . .	1144
ql/Math/ <a href="#">incompletegamma.hpp</a> (Incomplete Gamma function ) . . . . .	1145
ql/Math/ <a href="#">incrementalstatistics.hpp</a> (Statistics tool based on incremental accumulation ) .	1146
ql/Math/ <a href="#">interpolation.hpp</a> (Base class for 1-D interpolations ) . . . . .	1147
ql/Math/ <a href="#">interpolation2D.hpp</a> (Abstract base classes for 2-D interpolations ) . . . . .	1148
ql/Math/ <a href="#">kronrodintegral.hpp</a> (Integral of a 1-dimensional function using the Gauss-Kronrod method ) . . . . .	1149
ql/Math/ <a href="#">lexicographicalview.hpp</a> (Lexicographical 2-D view of a contiguous set of data )	1150
ql/Math/ <a href="#">linearinterpolation.hpp</a> (Linear interpolation between discrete points ) . . . . .	1151
ql/Math/ <a href="#">loglinearinterpolation.hpp</a> (Log-linear interpolation between discrete points ) .	1152
ql/Math/ <a href="#">matrix.hpp</a> (Matrix used in linear algebra ) . . . . .	1153
ql/Math/ <a href="#">multicubicspline.hpp</a> (N-dimensional cubic spline interpolation between discrete points ) . . . . .	1154
ql/Math/ <a href="#">normaldistribution.hpp</a> (Normal, cumulative and inverse cumulative distributions ) . . . . .	1156
ql/Math/ <a href="#">poissondistribution.hpp</a> (Poisson distribution ) . . . . .	1157
ql/Math/ <a href="#">primenumbers.hpp</a> (Prime numbers calculator ) . . . . .	1158
ql/Math/ <a href="#">pseudosqrt.hpp</a> (Pseudo square root of a real symmetric matrix ) . . . . .	1159
ql/Math/ <a href="#">riskstatistics.hpp</a> (Empirical-distribution risk measures ) . . . . .	1160
ql/Math/ <a href="#">rounding.hpp</a> (Rounding implementation ) . . . . .	1161
ql/Math/ <a href="#">sampledcurve.hpp</a> (Class that contains a sampled curve ) . . . . .	1162
ql/Math/ <a href="#">segmentintegral.hpp</a> (Integral of a one-dimensional function ) . . . . .	1163
ql/Math/ <a href="#">sequencestatistics.hpp</a> (Statistics tools for sequence (vector, list, array) samples )	1164
ql/Math/ <a href="#">simpsonintegral.hpp</a> (Integral of a one-dimensional function ) . . . . .	1166
ql/Math/ <a href="#">statistics.hpp</a> (Statistics tool with risk measures ) . . . . .	1167
ql/Math/ <a href="#">svd.hpp</a> (Singular value decomposition ) . . . . .	1168
ql/Math/ <a href="#">symmetriceigenvalues.hpp</a> (Eigenvalues / eigenvectors of a real symmetric matrix ) . . . . .	1169
ql/Math/ <a href="#">symmetricschurdecomposition.hpp</a> (Eigenvalues / eigenvectors of a real symmetric matrix ) . . . . .	1170
ql/Math/ <a href="#">tqreigendecomposition.hpp</a> (Tridiag. QR eigen decomposition with explicit shift aka Wilkinson ) . . . . .	1171



ql/Math/ <a href="#">transformedgrid.hpp</a> (Encapsulates a grid ) . . . . .	1172
ql/Math/ <a href="#">trapezoidintegral.hpp</a> (Integral of a one-dimensional function ) . . . . .	1173
ql/MonteCarlo/ <a href="#">brownianbridge.hpp</a> (Browian bridge ) . . . . .	1175
ql/MonteCarlo/ <a href="#">getcovariance.hpp</a> (Covariance matrix calculation ) . . . . .	1176
ql/MonteCarlo/ <a href="#">mctraits.hpp</a> (Monte Carlo policies ) . . . . .	1177
ql/MonteCarlo/ <a href="#">mctypedefs.hpp</a> (Default choices for template instantiations ) . . . . .	1178
ql/MonteCarlo/ <a href="#">montecarlomodel.hpp</a> (General purpose Monte Carlo model ) . . . . .	1179
ql/MonteCarlo/ <a href="#">multipath.hpp</a> (Correlated multiple asset paths ) . . . . .	1180
ql/MonteCarlo/ <a href="#">multipathgenerator.hpp</a> (Generates a multi path from a random-array generator ) . . . . .	1181
ql/MonteCarlo/ <a href="#">path.hpp</a> (Single factor random walk ) . . . . .	1182
ql/MonteCarlo/ <a href="#">pathgenerator.hpp</a> (Generates random paths using a sequence generator ) . . . . .	1183
ql/MonteCarlo/ <a href="#">pathpricer.hpp</a> (Base class for single-path pricers ) . . . . .	1184
ql/MonteCarlo/ <a href="#">sample.hpp</a> (Weighted sample ) . . . . .	1185
ql/Optimization/ <a href="#">armijo.hpp</a> (Armijo line-search class ) . . . . .	1187
ql/Optimization/ <a href="#">conjugategradient.hpp</a> (Conjugate gradient optimization method ) . . . . .	1188
ql/Optimization/ <a href="#">constraint.hpp</a> (Abstract constraint class ) . . . . .	1189
ql/Optimization/ <a href="#">costfunction.hpp</a> (Optimization cost function class ) . . . . .	1190
ql/Optimization/ <a href="#">criteria.hpp</a> (Optimization criteria class ) . . . . .	1191
ql/Optimization/ <a href="#">leastsquare.hpp</a> (Least square cost function ) . . . . .	1192
ql/Optimization/ <a href="#">levenbergmarquardt.hpp</a> (Levenberg-Marquardt optimization method ) . . . . .	1193
ql/Optimization/ <a href="#">linesearch.hpp</a> (Line search abstract class ) . . . . .	1194
ql/Optimization/ <a href="#">lmdif.hpp</a> (Wrapper for MINPACK minimization routine ) . . . . .	1195
ql/Optimization/ <a href="#">method.hpp</a> (Abstract optimization method class ) . . . . .	1196
ql/Optimization/ <a href="#">problem.hpp</a> (Abstract optimization class ) . . . . .	1197
ql/Optimization/ <a href="#">simplex.hpp</a> (Simplex optimization method ) . . . . .	1198
ql/Optimization/ <a href="#">steepestdescent.hpp</a> (Steepest descent optimization method ) . . . . .	1199
ql/Patterns/ <a href="#">bridge.hpp</a> (Bridge pattern (a.k.a. handle-body idiom) ) . . . . .	1201
ql/Patterns/ <a href="#">composite.hpp</a> (Composite pattern ) . . . . .	1202
ql/Patterns/ <a href="#">curiouslyrecurring.hpp</a> (Curiously recurring template pattern ) . . . . .	1203
ql/Patterns/ <a href="#">lazyobject.hpp</a> (Framework for calculation on demand and result caching ) . . . . .	1204
ql/Patterns/ <a href="#">observable.hpp</a> (Observer/observable pattern ) . . . . .	1205
ql/Patterns/ <a href="#">singleton.hpp</a> (Basic support for the singleton pattern ) . . . . .	1206
ql/Patterns/ <a href="#">visitor.hpp</a> (Degenerate base class for the Acyclic Visitor pattern ) . . . . .	1207
ql/Pricers/ <a href="#">discretegeometriccaso.hpp</a> (Discrete Geometric Average Strike Option ) . . . . .	1209
ql/Pricers/ <a href="#">mccliquetoption.hpp</a> (Cliquet option priced with Monte Carlo simulation ) . . . . .	1210
ql/Pricers/ <a href="#">mcdiscretearithmeticcaso.hpp</a> (Discrete Arithmetic Average Strike Option ) . . . . .	1211
ql/Pricers/ <a href="#">mceverest.hpp</a> (Everest-type option pricer ) . . . . .	1212
ql/Pricers/ <a href="#">mchimalaya.hpp</a> (Himalayan-type option pricer ) . . . . .	1213
ql/Pricers/ <a href="#">mcmaxbasket.hpp</a> (Max Basket Monte Carlo pricer ) . . . . .	1214
ql/Pricers/ <a href="#">mcpagoda.hpp</a> (Roofed multi asset Asian option ) . . . . .	1215
ql/Pricers/ <a href="#">mcperformanceoption.hpp</a> (Performance option priced with Monte Carlo simulation ) . . . . .	1216
ql/Pricers/ <a href="#">mcpricer.hpp</a> (Base class for Monte Carlo pricers ) . . . . .	1217
ql/Pricers/ <a href="#">singleassetoption.hpp</a> (Common code for option evaluation ) . . . . .	1218
ql/PricingEngines/ <a href="#">americanpayoffatexpiry.hpp</a> (Analytical formulae for american exercise with payoff at expiry ) . . . . .	1220
ql/PricingEngines/ <a href="#">americanpayoffathit.hpp</a> (Analytical formulae for american exercise with payoff at hit ) . . . . .	1221
ql/PricingEngines/ <a href="#">blackformula.hpp</a> (Black formula ) . . . . .	1232
ql/PricingEngines/ <a href="#">blackmodel.hpp</a> (Abstract class for Black-type models (market models) ) . . . . .	1233
ql/PricingEngines/ <a href="#">genericmodelengine.hpp</a> (Generic option engine based on a model ) . . . . .	1242
ql/PricingEngines/ <a href="#">greeks.hpp</a> (Default greek calculations ) . . . . .	1243



ql/PricingEngines/ <a href="#">latticeshortratemodelengine.hpp</a> (Engine for a short-rate model specialized on a lattice ) . . . . .	1246
ql/PricingEngines/ <a href="#">mcsimulation.hpp</a> (Framework for Monte Carlo engines ) . . . . .	1247
ql/PricingEngines/Asian/ <a href="#">analytic_cont_geom_av_price.hpp</a> (Analytic engine for continuous geometric average price Asian ) . . . . .	1222
ql/PricingEngines/Asian/ <a href="#">analytic_discr_geom_av_price.hpp</a> (Analytic engine for discrete geometric average price Asian ) . . . . .	1223
ql/PricingEngines/Asian/ <a href="#">mc_discr_arith_av_price.hpp</a> (Monte Carlo engine for discrete arithmetic average price Asian ) . . . . .	1224
ql/PricingEngines/Asian/ <a href="#">mc_discr_geom_av_price.hpp</a> (Monte Carlo engine for discrete geometric average price Asian ) . . . . .	1225
ql/PricingEngines/Asian/ <a href="#">mcdiscreteasianengine.hpp</a> (Monte Carlo pricing engine for discrete average Asians ) . . . . .	1226
ql/PricingEngines/Barrier/ <a href="#">analyticbarrierengine.hpp</a> (Analytic barrier option engines ) . . . . .	1227
ql/PricingEngines/Barrier/ <a href="#">mcbarrierengine.hpp</a> (Monte Carlo barrier option engines ) . . . . .	1228
ql/PricingEngines/Basket/ <a href="#">mcamericanbasketengine.hpp</a> (Least-square Monte Carlo engines ) . . . . .	1229
ql/PricingEngines/Basket/ <a href="#">mcbasketengine.hpp</a> (European basket MC Engine ) . . . . .	1230
ql/PricingEngines/Basket/ <a href="#">stulzengine.hpp</a> (2D European Basket formulae, due to Stulz (1982) ) . . . . .	1231
ql/PricingEngines/CapFloor/ <a href="#">analyticcapfloorengine.hpp</a> (Analytic engine for caps/floors ) . . . . .	1234
ql/PricingEngines/CapFloor/ <a href="#">blackcapfloorengine.hpp</a> (Black-formula cap/floor engine ) . . . . .	1235
ql/PricingEngines/CapFloor/ <a href="#">discretizedcapfloor.hpp</a> (Discretized cap/floor ) . . . . .	1236
ql/PricingEngines/CapFloor/ <a href="#">treecapfloorengine.hpp</a> (Numerical lattice engine for cap/floors ) . . . . .	1237
ql/PricingEngines/Cliquet/ <a href="#">analyticcliquetengine.hpp</a> (Analytic Cliquet engine ) . . . . .	1238
ql/PricingEngines/Cliquet/ <a href="#">analyticperformanceengine.hpp</a> (Analytic performance engine ) . . . . .	1239
ql/PricingEngines/Forward/ <a href="#">forwardengine.hpp</a> (Forward (strike-resetting) option engine ) . . . . .	1240
ql/PricingEngines/Forward/ <a href="#">forwardperformanceengine.hpp</a> (Forward (strike-resetting) performance option engines ) . . . . .	1241
ql/PricingEngines/Hybrid/ <a href="#">binomialconvertibleengine.hpp</a> (Binomial engine for convertible bonds ) . . . . .	1244
ql/PricingEngines/Hybrid/ <a href="#">discretizedconvertible.hpp</a> (Discretized convertible ) . . . . .	1245
ql/PricingEngines/Quanto/ <a href="#">quantoengine.hpp</a> (Quanto option engine ) . . . . .	1248
ql/PricingEngines/Swapption/ <a href="#">blackswapptionengine.hpp</a> (Black-formula swapption engine ) . . . . .	1249
ql/PricingEngines/Swapption/ <a href="#">discretizedswapption.hpp</a> (Discretized swapption class ) . . . . .	1250
ql/PricingEngines/Swapption/ <a href="#">g2swapptionengine.hpp</a> (Swapption pricing engine for two-factor additive Gaussian Model G2++ ) . . . . .	1251
ql/PricingEngines/Swapption/ <a href="#">jamshidianswapptionengine.hpp</a> (Swapption engine using Jamshidian's decomposition ) . . . . .	1252
ql/PricingEngines/Swapption/ <a href="#">lfmswapptionengine.hpp</a> (Libor forward model swapption engine based on black formula ) . . . . .	1253
ql/PricingEngines/Swapption/ <a href="#">treeswapptionengine.hpp</a> (Numerical lattice engines for swaps and swapptions ) . . . . .	1254
ql/PricingEngines/Vanilla/ <a href="#">analyticdigitalamericanengine.hpp</a> (Analytic digital American option engine ) . . . . .	1255
ql/PricingEngines/Vanilla/ <a href="#">analyticdividendeuropeanengine.hpp</a> (Analytic discrete-dividend European engine ) . . . . .	1256
ql/PricingEngines/Vanilla/ <a href="#">analyticeuropeanengine.hpp</a> (Analytic European engine ) . . . . .	1257
ql/PricingEngines/Vanilla/ <a href="#">analytichestonengine.hpp</a> (Analytic Heston-model engine ) . . . . .	1258
ql/PricingEngines/Vanilla/ <a href="#">baroneadesiwhaleyengine.hpp</a> (Barone-Adesi and Whaley approximation engine ) . . . . .	1259

ql/PricingEngines/Vanilla/batesengine.hpp (Analytic Bates model engine ) . . . . .	1260
ql/PricingEngines/Vanilla/binomialengine.hpp (Binomial option engine ) . . . . .	1261
ql/PricingEngines/Vanilla/bjersundstenslandengine.hpp (Bjersund and Stensland approximation engine ) . . . . .	1262
ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp (Discretized vanilla option ) . .	1263
ql/PricingEngines/Vanilla/fdamericanengine.hpp (Finite-differences American option engine ) . . . . .	1264
ql/PricingEngines/Vanilla/fdbermudanengine.hpp (Finite-difference Bermudan engine )	1265
ql/PricingEngines/Vanilla/fdconditions.hpp (Finite-difference templates to generate engines ) . . . . .	1266
ql/PricingEngines/Vanilla/fddividendamericanengine.hpp (American engine with discrete deterministic dividends ) . . . . .	1267
ql/PricingEngines/Vanilla/fddividendengine.hpp (Base engine for option with dividends )	1268
ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp (Finite-differences engine for European option with dividends ) . . . . .	1269
ql/PricingEngines/Vanilla/fddividendshoutengine.hpp (Base class for shout engine with dividends ) . . . . .	1270
ql/PricingEngines/Vanilla/fdeuropeanengine.hpp (Finite-difference European engine ) .	1271
ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp (Base engine for options with events happening at specific times ) . . . . .	1272
ql/PricingEngines/Vanilla/fdshoutengine.hpp (Finite-differences shout engine ) . . . .	1273
ql/PricingEngines/Vanilla/fdstepconditionengine.hpp (Finite-differences step-condition engine ) . . . . .	1274
ql/PricingEngines/Vanilla/fdvanillaengine.hpp (Finite-differences vanilla-option engine )	1275
ql/PricingEngines/Vanilla/integralengine.hpp (Integral option engine ) . . . . .	1276
ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp (Jump diffusion (Merton 1976) engine ) . . . . .	1277
ql/PricingEngines/Vanilla/juquadraticengine.hpp (Ju quadratic (1999) approximation engine ) . . . . .	1278
ql/PricingEngines/Vanilla/mcdigitalengine.hpp (Digital option Monte Carlo engine ) . .	1279
ql/PricingEngines/Vanilla/mceuropeanengine.hpp (Monte Carlo European option engine ) . . . . .	1280
ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp (Monte Carlo Heston-model engine for European options ) . . . . .	1281
ql/PricingEngines/Vanilla/mcvanillaengine.hpp (Monte Carlo vanilla option engine ) . .	1282
ql/Processes/blackscholesprocess.hpp (Black-Scholes processes ) . . . . .	1283
ql/Processes/eulerdiscretization.hpp (Euler discretization for stochastic processes ) . .	1284
ql/Processes/geometricbrownianprocess.hpp (Geometric Brownian-motion process ) . .	1285
ql/Processes/hestonprocess.hpp (Heston stochastic process ) . . . . .	1286
ql/Processes/lfmcovarParams.hpp (Volatility & correlation function for libor forward model process ) . . . . .	1287
ql/Processes/lfmhullwhiteparam.hpp (Libor market model parameterization based on Hull White ) . . . . .	1288
ql/Processes/lfmprocess.hpp (Stochastic process of a libor forward model ) . . . . .	1289
ql/Processes/merton76process.hpp (Merton-76 process ) . . . . .	1290
ql/Processes/ornsteinuhlenbeckprocess.hpp (Ornstein-Uhlenbeck process ) . . . . .	1291
ql/Processes/squarerootprocess.hpp (Square-root process ) . . . . .	1292
ql/Processes/stochasticprocessarray.hpp (Array of correlated 1-D stochastic processes ) .	1293
ql/RandomNumbers/boxmullergaussianrng.hpp (Box-Muller Gaussian random-number generator ) . . . . .	1297
ql/RandomNumbers/centrallimitgaussianrng.hpp (Central limit Gaussian random-number generator ) . . . . .	1298
ql/RandomNumbers/faurersg.hpp (Faure low-discrepancy sequence generator ) . . . .	1299
ql/RandomNumbers/haltonrng.hpp (Halton low-discrepancy sequence generator ) . . .	1300

ql/RandomNumbers/ <a href="#">inversecumulativerng.hpp</a> (Inverse cumulative Gaussian random-number generator) . . . . .	1301
ql/RandomNumbers/ <a href="#">inversecumulativersg.hpp</a> (Inverse cumulative random sequence generator) . . . . .	1302
ql/RandomNumbers/ <a href="#">knuthuniformrng.hpp</a> (Knuth uniform random number generator) . . . . .	1303
ql/RandomNumbers/ <a href="#">lecuyeruniformrng.hpp</a> (L'Ecuyer uniform random number generator) . . . . .	1304
ql/RandomNumbers/ <a href="#">mt19937uniformrng.hpp</a> (Mersenne Twister uniform random number generator) . . . . .	1305
ql/RandomNumbers/ <a href="#">randomizedlds.hpp</a> (Randomized low-discrepancy sequence) . . . . .	1306
ql/RandomNumbers/ <a href="#">randomsequencegenerator.hpp</a> (Random sequence generator based on a pseudo-random number generator) . . . . .	1307
ql/RandomNumbers/ <a href="#">rngtraits.hpp</a> (Random-number generation policies) . . . . .	1308
ql/RandomNumbers/ <a href="#">seedgenerator.hpp</a> (Random seed generator) . . . . .	1310
ql/RandomNumbers/ <a href="#">sobolrsg.hpp</a> (Sobol low-discrepancy sequence generator) . . . . .	1311
ql/ShortRateModels/ <a href="#">calibrationhelper.hpp</a> (Calibration helper class) . . . . .	1314
ql/ShortRateModels/ <a href="#">model.hpp</a> (Abstract interest rate model class) . . . . .	1325
ql/ShortRateModels/ <a href="#">onefactormodel.hpp</a> (Abstract one-factor interest rate model class) . . . . .	1326
ql/ShortRateModels/ <a href="#">parameter.hpp</a> (Model parameter classes) . . . . .	1332
ql/ShortRateModels/ <a href="#">twofactormodel.hpp</a> (Abstract two-factor interest rate model class) . . . . .	1333
ql/ShortRateModels/CalibrationHelpers/ <a href="#">caphelper.hpp</a> (CapHelper calibration helper) . . . . .	1315
ql/ShortRateModels/CalibrationHelpers/ <a href="#">hestonmodelhelper.hpp</a> (Heston-model calibration helper) . . . . .	1316
ql/ShortRateModels/CalibrationHelpers/ <a href="#">swaptionhelper.hpp</a> (Swaption calibration helper) . . . . .	1317
ql/ShortRateModels/LiborMarketModels/ <a href="#">lfmcoverproxy.hpp</a> (Proxy for libor forward covariance parameterization) . . . . .	1318
ql/ShortRateModels/LiborMarketModels/ <a href="#">liborforwardmodel.hpp</a> (Libor forward model incl. exact cap pricing Rebonato formula to approximate swaption prices) . . . . .	1319
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmcormodel.hpp</a> (Correlation model for libor market models) . . . . .	1320
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmexpcormodel.hpp</a> (Exponential correlation model for libor market models) . . . . .	1321
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmfixedvolmodel.hpp</a> (Model of constant volatilities for libor market models) . . . . .	1322
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmlexpvolmodel.hpp</a> (Volatility model for libor market models) . . . . .	1323
ql/ShortRateModels/LiborMarketModels/ <a href="#">lmvolmodel.hpp</a> (Volatility model for libor market models) . . . . .	1324
ql/ShortRateModels/OneFactorModels/ <a href="#">blackkarasinski.hpp</a> (Black-Karasinski model) . . . . .	1327
ql/ShortRateModels/OneFactorModels/ <a href="#">coxingersollross.hpp</a> (Cox-Ingersoll-Ross model) . . . . .	1328
ql/ShortRateModels/OneFactorModels/ <a href="#">extendedcoxingersollross.hpp</a> (Extended Cox-Ingersoll-Ross model) . . . . .	1329
ql/ShortRateModels/OneFactorModels/ <a href="#">hullwhite.hpp</a> (Hull & White (HW) model) . . . . .	1330
ql/ShortRateModels/OneFactorModels/ <a href="#">vasicek.hpp</a> (Vasicek model class) . . . . .	1331
ql/ShortRateModels/TwoFactorModels/ <a href="#">batesmodel.hpp</a> (Extended versions of the Heston model) . . . . .	1334
ql/ShortRateModels/TwoFactorModels/ <a href="#">g2.hpp</a> (Two-factor additive Gaussian Model G2++) . . . . .	1335
ql/ShortRateModels/TwoFactorModels/ <a href="#">hestonmodel.hpp</a> (Heston model for the stochastic volatility of an asset) . . . . .	1336
ql/Solvers1D/ <a href="#">bisection.hpp</a> (Bisection 1-D solver) . . . . .	1338
ql/Solvers1D/ <a href="#">brent.hpp</a> (Brent 1-D solver) . . . . .	1339
ql/Solvers1D/ <a href="#">falseposition.hpp</a> (False-position 1-D solver) . . . . .	1340

ql/Solvers1D/newton.hpp (Newton 1-D solver) . . . . .	1341
ql/Solvers1D/newtonsafe.hpp (Safe (bracketed) Newton 1-D solver) . . . . .	1342
ql/Solvers1D/ridder.hpp (Ridder 1-D solver) . . . . .	1343
ql/Solvers1D/secant.hpp (Secant 1-D solver) . . . . .	1344
ql/TermStructures/affinetermstructure.hpp (Affine term structure) . . . . .	1348
ql/TermStructures/bondhelpers.hpp (Bond rate helpers) . . . . .	1349
ql/TermStructures/bootstraptraits.hpp (Bootstrap traits) . . . . .	1350
ql/TermStructures/compoundforward.hpp (Compounded forward term structure) . . . . .	1351
ql/TermStructures/discountcurve.hpp (Interpolated discount factor structure) . . . . .	1352
ql/TermStructures/drifttermstructure.hpp (Drift term structure) . . . . .	1353
ql/TermStructures/extendeddiscountcurve.hpp (Discount factor structure with detailed compound-forward calculation) . . . . .	1354
ql/TermStructures/flatforward.hpp (Flat forward rate term structure) . . . . .	1355
ql/TermStructures/forwardcurve.hpp (Interpolated forward-rate structure) . . . . .	1356
ql/TermStructures/forwardspreadedtermstructure.hpp (Forward-spreaded term struc- ture) . . . . .	1357
ql/TermStructures/forwardstructure.hpp (Forward-based yield term structure) . . . . .	1358
ql/TermStructures/impliedtermstructure.hpp (Implied term structure) . . . . .	1359
ql/TermStructures/piecewiseflatforward.hpp (Piecewise flat forward term structure) . . . . .	1360
ql/TermStructures/piecewiseyieldcurve.hpp (Piecewise-interpolated term structure) . . . . .	1361
ql/TermStructures/quantotermstructure.hpp (Quanto term structure) . . . . .	1362
ql/TermStructures/ratehelpers.hpp (Rate helpers base class) . . . . .	1363
ql/TermStructures/zerocurve.hpp (Interpolated zero-rates structure) . . . . .	1364
ql/TermStructures/zerospreadedtermstructure.hpp (Zero spreaded term structure) . . . . .	1365
ql/TermStructures/zeroyieldstructure.hpp (Zero-yield based term structure) . . . . .	1366
ql/Utilities/dataformatters.hpp (Output manipulators) . . . . .	1370
ql/Utilities/dataparsers.hpp (Classes used to parse data for input) . . . . .	1371
ql/Utilities/disposable.hpp (Generic disposable object with move semantics) . . . . .	1372
ql/Utilities/null.hpp (Null values) . . . . .	1373
ql/Utilities/observablevalue.hpp (Observable and assignable proxy to concrete value) . . . . .	1374
ql/Utilities/steppingiterator.hpp (Iterator advancing in constant steps) . . . . .	1375
ql/Utilities/strings.hpp (String utilities) . . . . .	1376
ql/Utilities/tracing.hpp (Tracing facilities) . . . . .	1377
ql/Volatilities/blackconstantvol.hpp (Black constant volatility, no time dependence, no strike dependence) . . . . .	1379
ql/Volatilities/blackvariancecurve.hpp (Black volatility curve modelled as variance curve) . . . . .	1380
ql/Volatilities/blackvariancesurface.hpp (Black volatility surface modelled as variance surface) . . . . .	1381
ql/Volatilities/capflatvolvector.hpp (Cap/floor at-the-money flat volatility vector) . . . . .	1382
ql/Volatilities/capletconstantvol.hpp (Constant caplet volatility) . . . . .	1383
ql/Volatilities/capletvariancecurve.hpp (Caplet variance curve) . . . . .	1384
ql/Volatilities/impliedvoltermstructure.hpp (Implied Black Vol Term Structure) . . . . .	1385
ql/Volatilities/localconstantvol.hpp (Local constant volatility, no time dependence, no asset dependence) . . . . .	1386
ql/Volatilities/localvolcurve.hpp (Local volatility curve derived from a Black curve) . . . . .	1387
ql/Volatilities/localvolsurface.hpp (Local volatility surface derived from a Black vol sur- face) . . . . .	1388
ql/Volatilities/swaptionvolmatrix.hpp (Swaption at-the-money volatility matrix) . . . . .	1389

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

- typedef [Real QuantLib::Spread](#)  
*spreads on interest rates*
- typedef [Real QuantLib::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 [MTCurrency](#)  
*Maltese lira.*
- class [NOKCurrency](#)  
*Norwegian krone.*
- class [PLNCurrency](#)  
*Polish zloty.*
- class [ROLCurrency](#)  
*Romanian 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.*
- 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 `Period`  
*Time period described by a number of a given time unit.*
- class `Date`  
*Concrete date class.*
- class `DayCounter`  
*day counter class*

### Typedefs

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

## Enumerations

- enum [QuantLib::BusinessDayConvention](#) {  
[QuantLib::Unadjusted](#), [QuantLib::Preceding](#), [QuantLib::ModifiedPreceding](#), [QuantLib::Following](#),  
[QuantLib::ModifiedFollowing](#), [QuantLib::MonthEndReference](#) }  
*Business Day conventions.*
- enum [QuantLib::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 [QuantLib::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 [QuantLib::Frequency](#) {  
[QuantLib::NoFrequency](#) = -1, [QuantLib::Once](#) = 0, [QuantLib::Annual](#) = 1, [QuantLib::Semiannual](#) = 2,  
[QuantLib::EveryFourthMonth](#) = 3, [QuantLib::Quarterly](#) = 4, [QuantLib::Bimonthly](#) = 6,  
[QuantLib::Monthly](#) = 12 }  
*Frequency of events.*
- enum [QuantLib::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:

*Unadjusted* Do not adjust.

*Preceding* Choose the first business day before the given holiday.

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

*Following* Choose the first business day after the given holiday.

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

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

**Examples:**

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

#### 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

**Examples:**

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

## 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 [China](#)  
*Chinese calendar.*
- class [India](#)  
*Indian calendars.*
- class [Slovakia](#)  
*Slovak calendars.*
- class [Brazil](#)  
*Brazilian calendar.*
- class [Hungary](#)  
*Hungarian calendar.*
- class [Denmark](#)  
*Danish calendar.*
- class [Germany](#)  
*German calendars.*
- class [Finland](#)  
*Finnish calendar.*
- class [HongKong](#)  
*Hong Kong calendars.*
- class [Iceland](#)  
*Icelandic calendars.*
- class [Indonesia](#)  
*Indonesian calendars*
- class [Turkey](#)  
*Turkish calendar.*

- class [Italy](#)  
*Italian calendars.*
- class [SouthAfrica](#)  
*South-African calendar.*
- class [JointCalendar](#)  
*Joint calendar.*
- class [Mexico](#)  
*Mexican calendars*
- class [NullCalendar](#)  
*Calendar for reproducing theoretical calculations.*
- class [Norway](#)  
*Norwegian calendar.*
- class [CzechRepublic](#)  
*Czech calendars.*
- class [SaudiArabia](#)  
*Saudi Arabian calendar.*
- class [SouthKorea](#)  
*South Korean calendars.*
- class [Singapore](#)  
*Singapore calendars*
- class [Sweden](#)  
*Swedish calendar.*
- class [Australia](#)  
*Australian calendar.*
- class [Taiwan](#)  
*Taiwanese calendars.*
- class [TARGET](#)  
*TARGET calendar*
- class [Japan](#)  
*Japanese calendar.*
- class [Canada](#)  
*Canadian calendar.*
- class [Ukraine](#)

*Ukrainian calendars.*

- class [UnitedKingdom](#)  
*United Kingdom calendars.*
- class [UnitedStates](#)  
*United States calendars.*
- class [Poland](#)  
*Polish calendar.*
- class [NewZealand](#)  
*New Zealand calendar.*
- class [Switzerland](#)  
*Swiss calendar.*



## 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 `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 [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.*



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

**Warning: this section of the documentation is currently outdated.** You will need to compare the information on this page with the present code for working pricers, such as `FdAmericanOption`.

This framework (corresponding to the `ql/FiniteDifferences` directory) contains basic building blocks for the numerical solution of a generic differential equation

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

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

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

### 6.16.2 Differential operators

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

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

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

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

and implemented in class `QuantLib::DPlus`; the operator  $D_-$ , defined as

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

and implemented in class `QuantLib::DMinus`; and the operator  $D_0$ , defined as

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

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

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

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

and implemented in class [QuantLib::DPlusDMinus](#). Its discretization error is  $O(h^2)$ .

The boundary condition for the above operators is by default linear extrapolation. Methods are currently provided for setting other kinds of boundary conditions to a tridiagonal operator which these operators inherit, namely, Dirichlet—i.e., constant value—and Neumann—i.e., constant derivative—boundary conditions. This might change in the future as boundary conditions could be abstracted and passed as an additional argument to the model.

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

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

### 6.16.3 Time schemes

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

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

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

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

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

hence

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

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

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

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

hence

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

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

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

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

hence

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

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

Each of the above evolvers forces a set of interface requirements upon the differential operator which are detailed in the documentation of the corresponding class, namely, [QuantLib::ExplicitEuler](#), [QuantLib::ImplicitEuler](#), and [QuantLib::CrankNicolson](#), respectively.

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

However, it must implement the following interface:

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

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

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

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

#### 6.16.4 Step conditions

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

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

#### 6.16.5 An example of finite difference model

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

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

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

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

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

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

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



```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

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

```
unsigned int timeSteps = 365;

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

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

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

## 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 > QuantLib::BSMTermOperator`  
*Black-Scholes-Merton differential operator.*
- `typedef PdeOperator< PdeShortRate > QuantLib::OneFactorOperator`  
*Interest-rate single factor model differential operator.*

### 6.16.6 Typedef Documentation

#### 6.16.6.1 `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-independant 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

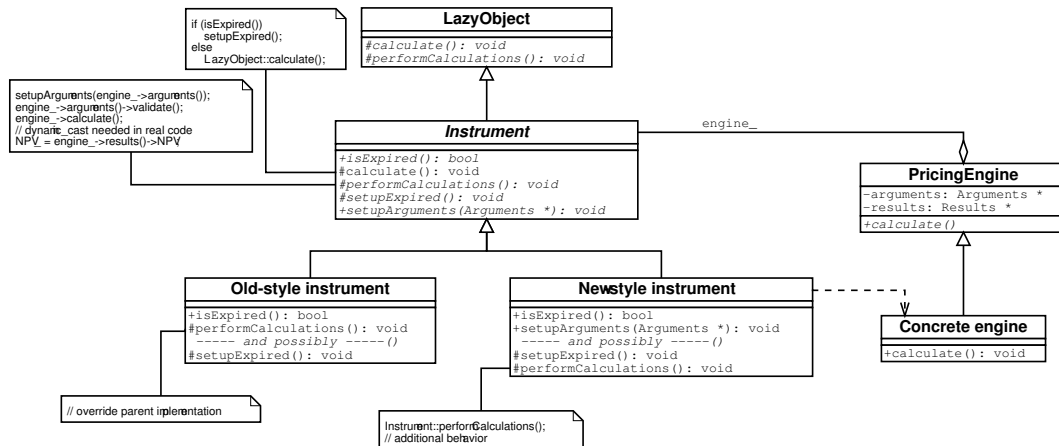
- class [AffineModel](#)  
*Affine model class.*
- class [TermStructureConsistentModel](#)  
*Term-structure consistent model class.*
- class [ShortRateModel](#)  
*Abstract short-rate model class.*
- class [OneFactorModel](#)  
*Single-factor short-rate model abstract class.*

- 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 [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 [DividendVanillaOption](#)  
*Single-asset vanilla option (no barriers) with discrete dividends.*
- class [EuropeanOption](#)  
*European option on a single asset.*

- class [FixedCouponBond](#)  
*fixed-coupon bond*
- class [FloatingRateBond](#)  
*floating-rate bond*
- class [ForwardVanillaOption](#)  
*Forward version of a vanilla option.*
- class [QuantoForwardVanillaOption](#)  
*Quanto version of a forward vanilla option.*
- class [QuantoVanillaOption](#)  
*quanto version of a vanilla option*
- class [VanillaSwap](#)  
*Plain-vanilla swap.*
- 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 [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  $P_0$  and  $N$  points such that

$$P_i = P_0 + \lambda e_i,$$

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

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

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

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

**Warning:** this section of the documentation is currently outdated.

This framework (corresponding to the `ql/MonteCarlo` directory) contains basic building blocks for the numerical calculation of the integral

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

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

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

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

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

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

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

### 6.21.2 Path generation

The Black-Scholes equation

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

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

Therefore, the paths to be generated for a Monte Carlo model of the Black-Scholes equation will be vectors of successive variations of the logarithm of the underlying price over  $M$  consecutive time intervals  $\Delta t_i, i = 0 \dots M-1$ . Each such variation will be drawn from a Gaussian distribution with average  $v\Delta T_i$  and standard deviation  $\sigma\sqrt{\Delta T_i}$  — or possibly  $v_i\Delta T_i$  and  $\sigma_i\sqrt{\Delta T_i}$  should  $v$  and  $\sigma$  vary in time.



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

The `QuantLib::MultiPath` class is a straightforward extension which acts as a vector of Path objects.

Classes are provided which generate paths and multi-paths with the desired drift and diffusion components, namely, `QuantLib::PathGenerator` and `QuantLib::MultiPathGenerator`.

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

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

### 6.21.3 Pricing an instrument on a path

The `QuantLib::PathPricer` class is the base class from which path pricers must inherit. The only method which subclasses are required to implement is

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

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

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

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

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

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

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

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

A number of path pricers are available in the library and listed in reference manual.

### 6.21.4 Accumulating and averaging samples

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

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

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

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

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

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

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

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

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

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

A [QuantLib::McPricer](#) class is also available which wraps the typical usage of a Monte Carlo model.

Details on the Monte Carlo Pricer interface will be available in the [Pricers](#) section.

### 6.21.5 Examples of Monte Carlo models

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

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

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

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

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

The corresponding path pricer can be implemented as:

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

        size_t n = path.size();

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

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

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

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

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

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

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

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

```
// stored parameters
...
};
```

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

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

The path generator can be instantiated as

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

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

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

The path pricer is instantiated as

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

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

The model can now be created and used as following:

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

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

model.addSamples(samples);

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

More examples of path pricers can be found in the `ql/MonteCarlo` directory, while examples of more sophisticated pricers which uses them in Monte Carlo models can be found in the `ql/Pricers` directory.

### 6.21.6 Notes

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

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

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

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

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

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

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

### Classes

- class [BrownianBridge](#)  
*Builds Wiener process paths using Gaussian variates.*
- 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 Term structures

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

### 6.23.2 Typedef Documentation

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

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

Log-linear interpolation guarantees piecewise-constant forward rates.



## 6.24 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  $[x_0, x_m, x_{2m}, \dots]$

## 6.25 QuantLib macros

### 6.25.1 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 "0.3.12"`  
*version string*
- `#define QL_HEX_VERSION 0x000312f0`  
*version hexadecimal number*
- `#define QL_LIB_VERSION "0_3_12"`  
*version string for output lib name*

## 6.26 Generic macros

### 6.26.1 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.*

### 6.26.2 Define Documentation

#### 6.26.2.1 `#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.

#### 6.26.2.2 `#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](#), and [swapvaluation.cpp](#).

## 6.27 Numeric limits

### 6.27.1 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)())`

### 6.27.2 Define Documentation

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

Defines the value of the largest representable negative integer value

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

Defines the value of the largest representable integer value

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

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

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

Defines the value of the smallest representable positive double value

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

Defines the value of the largest representable floating-point value

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

Defines the machine precision for operations over doubles

## 6.28 Template capabilities

### 6.28.1 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`

### 6.28.2 Define Documentation

#### 6.28.2.1 `#define QL_TYPENAME typename`

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

## 6.29 Iterator support

### 6.29.1 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`

### 6.29.2 Define Documentation

#### 6.29.2.1 `#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.

## 6.30 Output manipulators

### 6.30.1 Detailed Description

Helper functions for creating formatted output.

#### Functions

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



## 6.31 Debugging macros

### 6.31.1 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*

### 6.31.2 Define Documentation

#### 6.31.2.1 `#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](#).

### 6.31.2.2 **#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.

### 6.31.2.3 **#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.

### 6.31.2.4 **#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](#).

### 6.31.2.5 **#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](#).

#### 6.31.2.6 **#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](#).

#### 6.31.2.7 **#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](#).

#### 6.31.2.8 **#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](#).



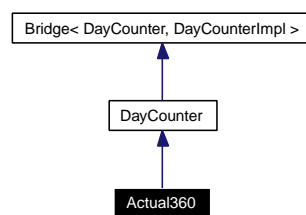
## Chapter 7

# QuantLib Class Documentation

### 7.1 Actual360 Class Reference

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

Inheritance diagram for Actual360:



#### 7.1.1 Detailed Description

Actual/360 day count convention.

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

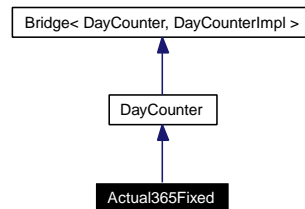
**Examples:**

[swapvaluation.cpp](#).

## 7.2 Actual365Fixed Class Reference

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

Inheritance diagram for Actual365Fixed:



### 7.2.1 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.

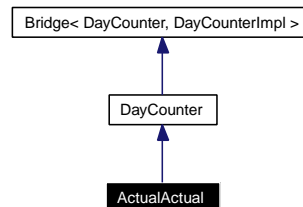
#### Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), and [Equity-Option.cpp](#).

## 7.3 ActualActual Class Reference

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

Inheritance diagram for ActualActual:



### 7.3.1 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:

[swapvaluation.cpp](#).

### Public Types

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

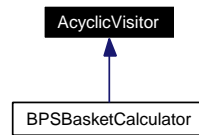
### Public Member Functions

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

## 7.4 AcyclicVisitor Class Reference

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

Inheritance diagram for AcyclicVisitor:



### 7.4.1 Detailed Description

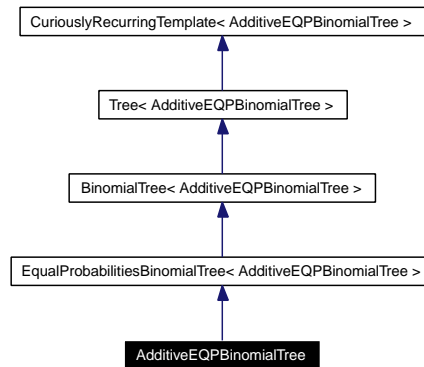
degenerate base class for the Acyclic Visitor pattern



## 7.5 AdditiveEQPBinoialTree Class Reference

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

Inheritance diagram for AdditiveEQPBinoialTree:



### 7.5.1 Detailed Description

Additive equal probabilities binomial tree.

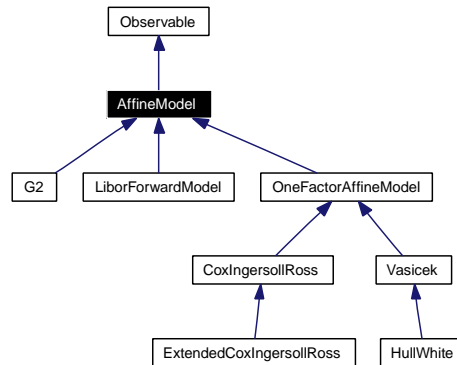
#### Public Member Functions

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

## 7.6 AffineModel Class Reference

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

Inheritance diagram for AffineModel:



### 7.6.1 Detailed Description

Affine model class.

Base class for analytically tractable models.

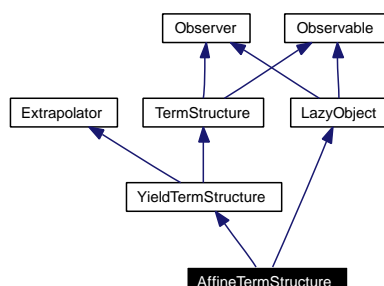
#### Public Member Functions

- 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

## 7.7 AffineTermStructure Class Reference

```
#include <ql/TermStructures/affinetermstructure.hpp>
```

Inheritance diagram for AffineTermStructure:



### 7.7.1 Detailed Description

Term-structure implied by an affine model.

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

### Public Member Functions

- [AffineTermStructure](#) (const [Date](#) &referenceDate, const boost::shared\_ptr< [AffineModel](#) > &model, const [DayCounter](#) &dayCounter)  
*constructor using a fixed model*
- [AffineTermStructure](#) (const [Date](#) &referenceDate, const boost::shared\_ptr< [AffineModel](#) > &model, const std::vector< boost::shared\_ptr< [RateHelper](#) > > &, const boost::shared\_ptr< [OptimizationMethod](#) > &, const [DayCounter](#) &dayCounter)  
*constructor using a model that has to be calibrated*
- [AffineTermStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &calendar, const boost::shared\_ptr< [AffineModel](#) > &model, const [DayCounter](#) &dayCounter)  
*constructor using a fixed model*
- [AffineTermStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &calendar, const boost::shared\_ptr< [AffineModel](#) > &model, const std::vector< boost::shared\_ptr< [RateHelper](#) > > &, const boost::shared\_ptr< [OptimizationMethod](#) > &, const [DayCounter](#) &dayCounter)  
*constructor using a model that has to be calibrated*
- [DayCounter](#) dayCounter () const  
*the day counter used for date/time conversion*

- [Date](#) `maxDate` () const  
*the latest date for which the curve can return rates*
- void [update](#) ()

## Protected Member Functions

- [DiscountFactor](#) `discountImpl` ([Time](#)) const  
*discount calculation*

## 7.7.2 Member Function Documentation

### 7.7.2.1 void `update` () [virtual]

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

Reimplemented from [LazyObject](#).

## 7.8 AmericanCondition Class Reference

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

### 7.8.1 Detailed Description

American exercise condition.

#### Todo

unify the intrinsicValues/Payoff thing

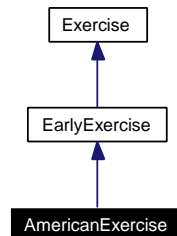
### Public Member Functions

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

## 7.9 AmericanExercise Class Reference

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

Inheritance diagram for AmericanExercise:



### 7.9.1 Detailed Description

American exercise.

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

#### 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)

## 7.10 AmericanPayoffAtExpiry Class Reference

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

### 7.10.1 Detailed Description

Analytic formula for American exercise payoff at-expiry options

#### Todo

calculate greeks

### Public Member Functions

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

## 7.11 AmericanPayoffAtHit Class Reference

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

### 7.11.1 Detailed Description

Analytic formula for American exercise payoff at-hit options

#### Todo

calculate greeks

### Public Member Functions

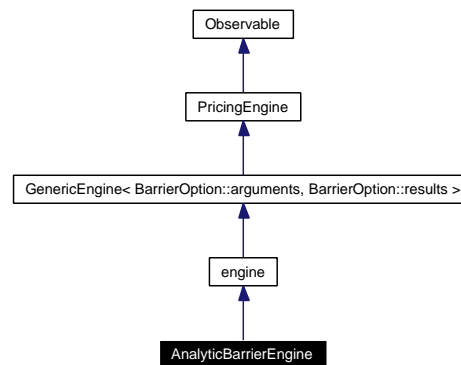
- **AmericanPayoffAtHit** ([Real](#) spot, [DiscountFactor](#) discount, [DiscountFactor](#) dividend-Discout, [Real](#) variance, const boost::shared\_ptr< [StrikedTypePayoff](#) > &payoff)
- [Real](#) **value** () const
- [Real](#) **delta** () const
- [Real](#) **gamma** () const
- [Real](#) **rho**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



## 7.12 AnalyticBarrierEngine Class Reference

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

Inheritance diagram for AnalyticBarrierEngine:



### 7.12.1 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

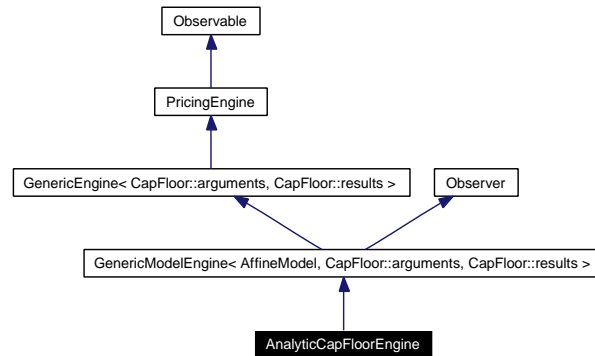
### Public Member Functions

- void **calculate** () const

## 7.13 AnalyticCapFloorEngine Class Reference

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

Inheritance diagram for AnalyticCapFloorEngine:



### 7.13.1 Detailed Description

Analytic engine for cap/floor.

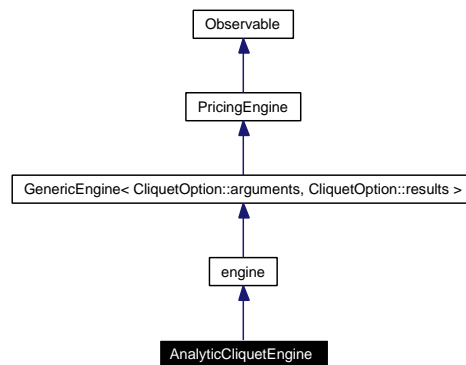
#### Public Member Functions

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

## 7.14 AnalyticCliquetEngine Class Reference

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

Inheritance diagram for AnalyticCliquetEngine:



### 7.14.1 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.

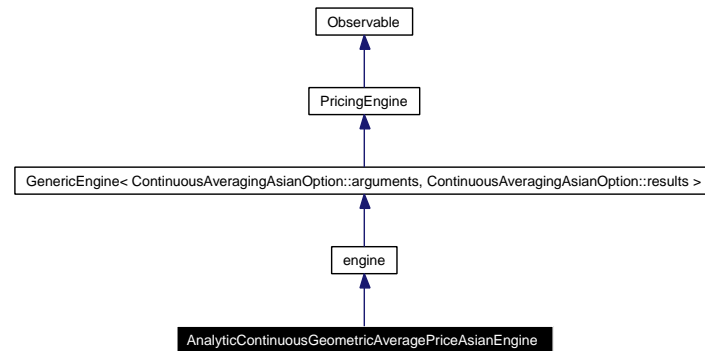
### Public Member Functions

- void **calculate** () const

## 7.15 AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference

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

Inheritance diagram for AnalyticContinuousGeometricAveragePriceAsianEngine:



### 7.15.1 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

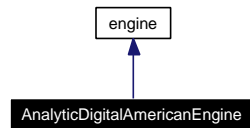
### Public Member Functions

- void **calculate** () const

## 7.16 AnalyticDigitalAmericanEngine Class Reference

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

Inheritance diagram for AnalyticDigitalAmericanEngine:



### 7.16.1 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.

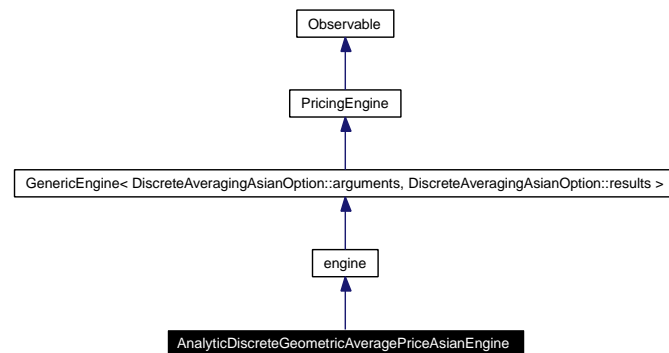
### Public Member Functions

- void **calculate** () const

## 7.17 AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference

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

Inheritance diagram for AnalyticDiscreteGeometricAveragePriceAsianEngine:



### 7.17.1 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.

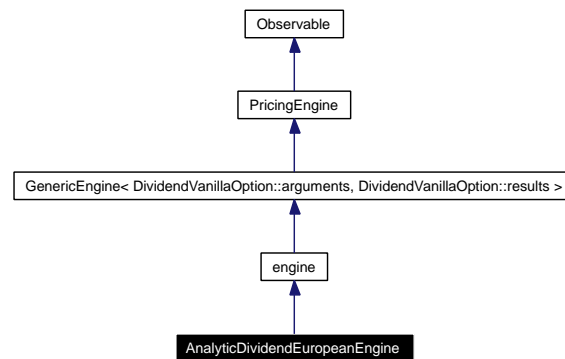
### Public Member Functions

- void **calculate** () const

## 7.18 AnalyticDividendEuropeanEngine Class Reference

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

Inheritance diagram for AnalyticDividendEuropeanEngine:



### 7.18.1 Detailed Description

Analytic pricing engine for European options with discrete dividends.

#### Tests

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

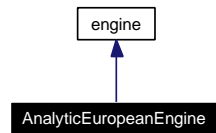
### Public Member Functions

- `void calculate () const`

## 7.19 AnalyticEuropeanEngine Class Reference

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

Inheritance diagram for AnalyticEuropeanEngine:



### 7.19.1 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

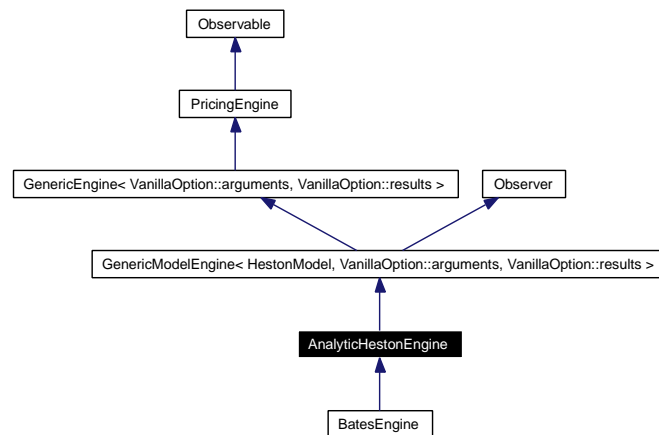
- void **calculate** () const



## 7.20 AnalyticHestonEngine Class Reference

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

Inheritance diagram for AnalyticHestonEngine:



### 7.20.1 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.

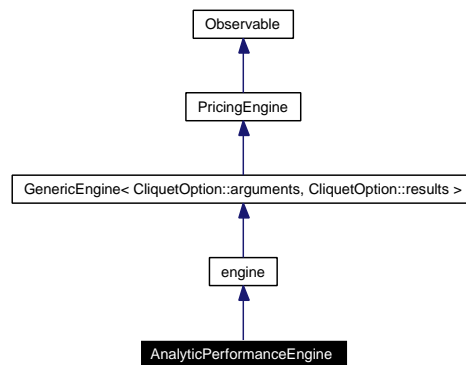
### Public Member Functions

- **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

## 7.21 AnalyticPerformanceEngine Class Reference

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

Inheritance diagram for AnalyticPerformanceEngine:



### 7.21.1 Detailed Description

Pricing engine for performance options using analytical formulae.

#### Tests

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

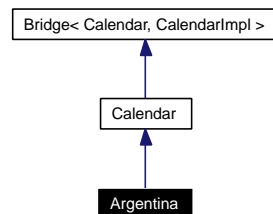
### Public Member Functions

- void **calculate** () const

## 7.22 Argentina Class Reference

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

Inheritance diagram for Argentina:



### 7.22.1 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)

## 7.22.2 Member Enumeration Documentation

### 7.22.2.1 enum [Market](#)

Enumerator:

*Merval* Buenos Aires stock exchange calendar.

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

```

graph BT
    A[arguments] --> B[arguments]
    B --> C[arguments]
    B --> D[arguments]
    B --> E[arguments]
    C --> F[Arguments]
    D --> F
    E --> F
  
```

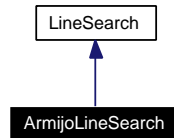
## base class for generic argument groups

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

## 7.24 ArmijoLineSearch Class Reference

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

Inheritance diagram for ArmijoLineSearch:



### 7.24.1 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.*
- virtual [Real operator\(\)](#) (const [Problem](#) &P, [Real](#) t\_ini)  
*Perform line search.*

## 7.25 Array Class Reference

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

### 7.25.1 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

### 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

#### 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 > &)`

#### 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](#) & **operator**[ ] ([Size](#))  
*read-write*

### 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](#) &)

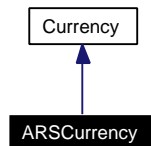


- const Disposable< Array > **operator**/(const Array &, const Array &)
- const Disposable< Array > **operator**/(const Array &, Real)
- const Disposable< Array > **operator**/(Real, const Array &)
- const Disposable< Array > **Abs** (const Array &)
- const Disposable< Array > **Sqrt** (const Array &)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

## 7.26 ARSCurrency Class Reference

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

Inheritance diagram for ARSCurrency:



### 7.26.1 Detailed Description

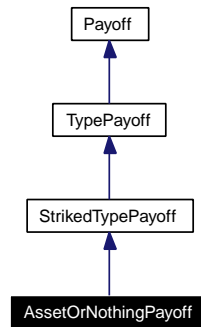
Argentinian peso.

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

## 7.27 AssetOrNothingPayoff Class Reference

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

Inheritance diagram for AssetOrNothingPayoff:



### 7.27.1 Detailed Description

Binary asset-or-nothing payoff.

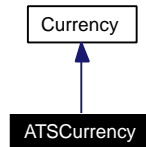
#### Public Member Functions

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

## 7.28 ATSCurrency Class Reference

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

Inheritance diagram for ATSCurrency:



### 7.28.1 Detailed Description

Austrian shilling.

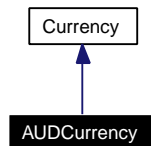
The ISO three-letter code was ATS; the numeric code was 40. It was divided in 100 groschen.

Obsoleted by the Euro since 1999.

## 7.29 AUDCurrency Class Reference

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

Inheritance diagram for AUDCurrency:



### 7.29.1 Detailed Description

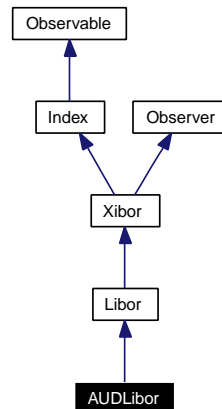
Australian dollar.

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

## 7.30 AUDLibor Class Reference

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

Inheritance diagram for AUDLibor:



### 7.30.1 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>>.

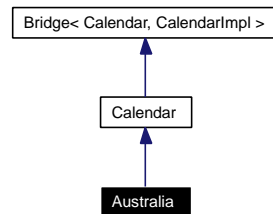
### Public Member Functions

- **AUDLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.31 Australia Class Reference

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

Inheritance diagram for Australia:



### 7.31.1 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)

## 7.32 Average Struct Reference

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

### 7.32.1 Detailed Description

placeholder for enumerated averaging types

#### Public Types

- enum Type { Arithmetic, Geometric }



## 7.33 BackwardFlat Class Reference

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

### 7.33.1 Detailed Description

Backward-flat interpolation factory and traits.

#### Public Types

- enum { **global** = 0 }

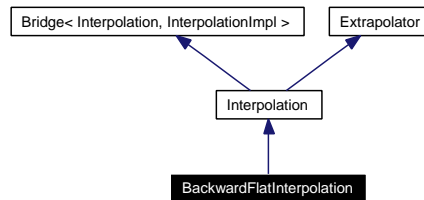
#### Public Member Functions

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

## 7.34 BackwardFlatInterpolation Class Reference

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

Inheritance diagram for BackwardFlatInterpolation:



### 7.34.1 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)`

### 7.34.2 Constructor & Destructor Documentation

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

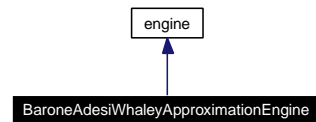
**Precondition:**

the *x* values must be sorted.

## 7.35 BaroneAdesiWhaleyApproximationEngine Class Reference

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

Inheritance diagram for BaroneAdesiWhaleyApproximationEngine:



### 7.35.1 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)

## 7.36 Barrier Struct Reference

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

### 7.36.1 Detailed Description

Placeholder for enumerated barrier types.

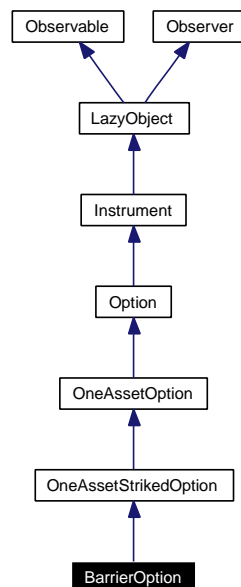
#### Public Types

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

## 7.37 BarrierOption Class Reference

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

Inheritance diagram for BarrierOption:



### 7.37.1 Detailed Description

Barrier option on a single asset.

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

#### 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*

## 7.37.2 Member Function Documentation

### 7.37.2.1 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](#).

## 7.38 BarrierOption::arguments Class Reference

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

### 7.38.1 Detailed Description

Arguments for barrier option calculation

#### Public Member Functions

- void **validate** () const

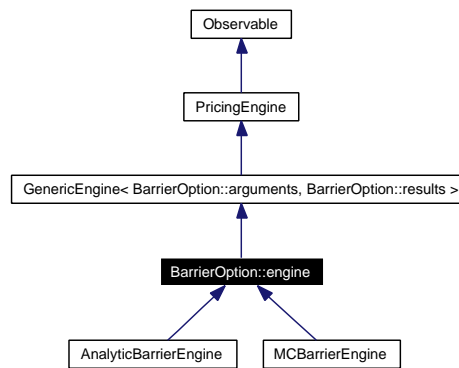
#### Public Attributes

- Barrier::Type **barrierType**
- [Real](#) **barrier**
- [Real](#) **rebate**

## 7.39 BarrierOption::engine Class Reference

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

Inheritance diagram for BarrierOption::engine:



### 7.39.1 Detailed Description

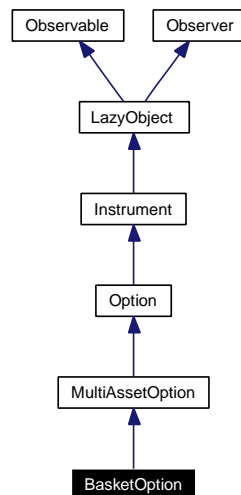
Barrier engine base class



## 7.40 BasketOption Class Reference

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

Inheritance diagram for BasketOption:



### 7.40.1 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*

## 7.40.2 Member Function Documentation

### 7.40.2.1 void setupArguments ([Arguments \\*](#)) const [virtual]

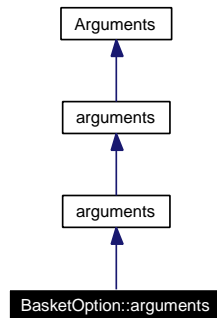
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [MultiAssetOption](#).

## 7.41 BasketOption::arguments Class Reference

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

Inheritance diagram for BasketOption::arguments:



### 7.41.1 Detailed Description

Arguments for basket option calculation

#### Public Member Functions

- void **validate** () const

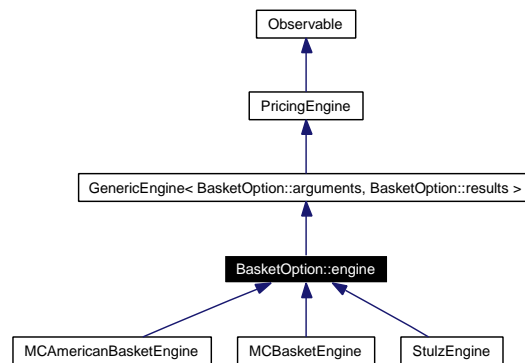
#### Public Attributes

- BasketType **basketType**

## 7.42 BasketOption::engine Class Reference

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

Inheritance diagram for BasketOption::engine:



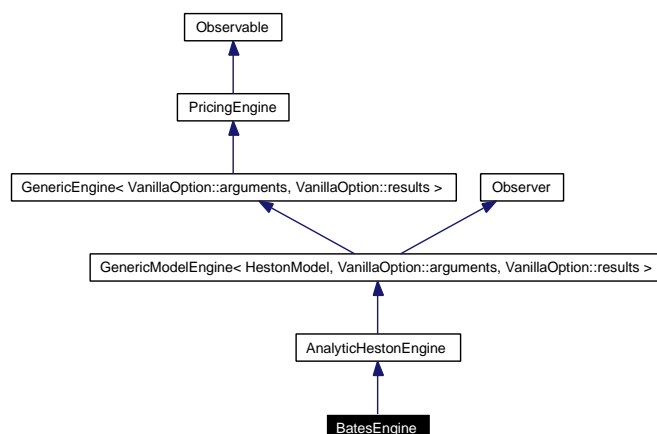
### 7.42.1 Detailed Description

Basket option engine base class

## 7.43 BatesEngine Class Reference

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

Inheritance diagram for BatesEngine:



### 7.43.1 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)

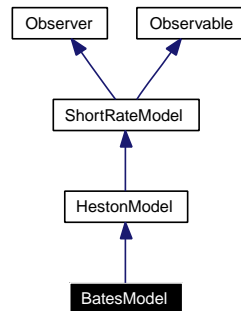
## Protected Member Functions

- std::complex< [Real](#) > **jumpDiffusionTerm** ([Real](#) phi, [Time](#) t, [Size](#) j) const

## 7.44 BatesModel Class Reference

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

Inheritance diagram for BatesModel:



### 7.44.1 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.

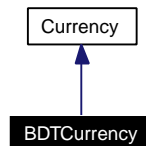
### Public Member Functions

- **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

## 7.45 BDTCurrency Class Reference

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

Inheritance diagram for BDTCurrency:



### 7.45.1 Detailed Description

Bangladesh taka.

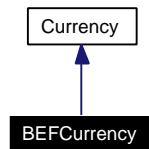
The ISO three-letter code is BDT; the numeric code is 50. It is divided in 100 paisa.



## 7.46 BEFCurrency Class Reference

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

Inheritance diagram for BEFCurrency:



### 7.46.1 Detailed Description

Belgian franc.

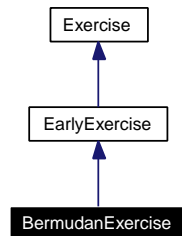
The ISO three-letter code was BEF; the numeric code was 56. It had no subdivisions.

Obsoleted by the Euro since 1999.

## 7.47 BermudanExercise Class Reference

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

Inheritance diagram for BermudanExercise:



### 7.47.1 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](#).

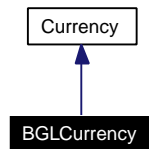
### Public Member Functions

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

## 7.48 BGLCurrency Class Reference

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

Inheritance diagram for BGLCurrency:



### 7.48.1 Detailed Description

Bulgarian lev.

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

## 7.49 Bicubic Class Reference

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

### 7.49.1 Detailed Description

bicubic-spline interpolation factory

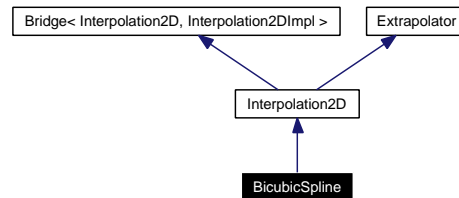
#### Public Member Functions

- `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`

## 7.50 BicubicSpline Class Reference

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

Inheritance diagram for BicubicSpline:



### 7.50.1 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)

### 7.50.2 Constructor & Destructor Documentation

- #### 7.50.2.1 [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.

## 7.51 Bilinear Class Reference

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

### 7.51.1 Detailed Description

bilinear interpolation factory

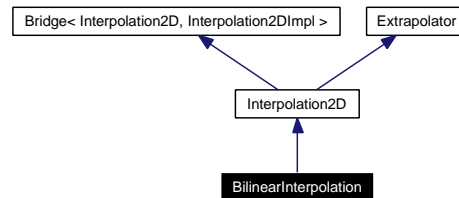
#### Public Member Functions

- `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`

## 7.52 BilinearInterpolation Class Reference

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

Inheritance diagram for BilinearInterpolation:



### 7.52.1 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)`

### 7.52.2 Constructor & Destructor Documentation

- 7.52.2.1 `BilinearInterpolation (const I1 & xBegin, const I1 & xEnd, const I2 & yBegin, const I2 & yEnd, const M & zData)`

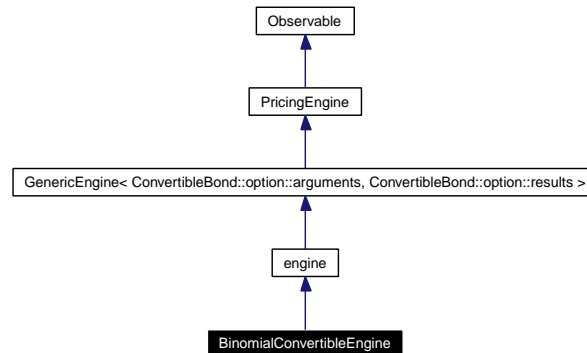
**Precondition:**

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

## 7.53 BinomialConvertibleEngine Class Template Reference

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

Inheritance diagram for BinomialConvertibleEngine:



### 7.53.1 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



## 7.54 BinomialDistribution Class Reference

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

### 7.54.1 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.

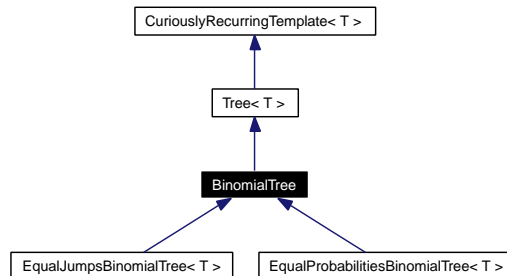
### Public Member Functions

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

## 7.55 BinomialTree Class Template Reference

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

Inheritance diagram for BinomialTree:



### 7.55.1 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

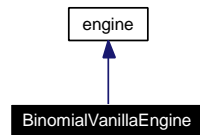
#### Protected Attributes

- [Real](#) x0\_
- [Real](#) driftPerStep\_
- [Time](#) dt\_

## 7.56 BinomialVanillaEngine Class Template Reference

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

Inheritance diagram for BinomialVanillaEngine:



### 7.56.1 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](#).

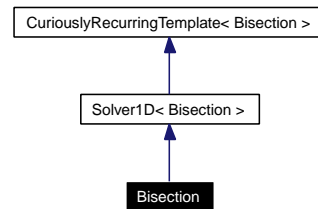
### Public Member Functions

- `BinomialVanillaEngine` ([Size](#) timeSteps)
- `void calculate () const`

## 7.57 Bisection Class Reference

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

Inheritance diagram for Bisection:



### 7.57.1 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`

## 7.58 BivariateCumulativeNormalDistributionDr78 Class Reference

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

### 7.58.1 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

## 7.59 BivariateCumulativeNormalDistributionWe04DP Class Reference

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

### 7.59.1 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.

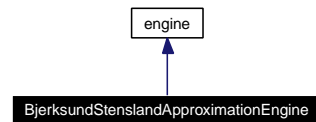
### Public Member Functions

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

## 7.60 Bjerk SundStenslandApproximationEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/bjerk SundStenslandengine.hpp>
```

Inheritance diagram for Bjerk SundStenslandApproximationEngine:



### 7.60.1 Detailed Description

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

#### Tests

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

#### Examples:

[EquityOption.cpp](#).

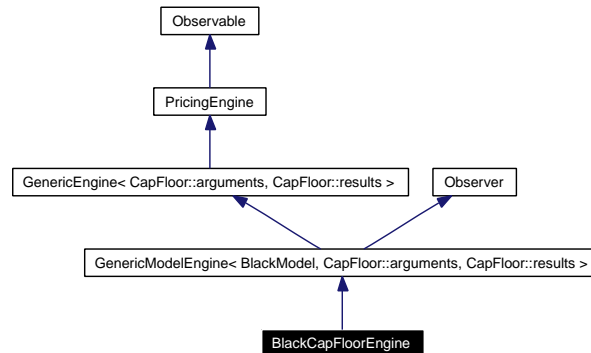
### Public Member Functions

- void **calculate** () const

## 7.61 BlackCapFloorEngine Class Reference

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

Inheritance diagram for BlackCapFloorEngine:



### 7.61.1 Detailed Description

Black-formula cap/floor engine.

#### Public Member Functions

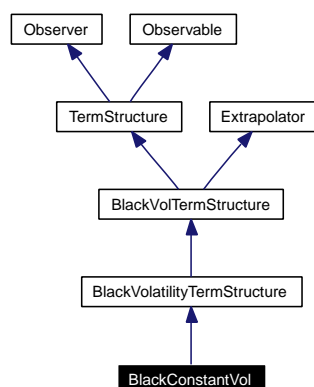
- **BlackCapFloorEngine** (const boost::shared\_ptr< [BlackModel](#) > &model)
- void **calculate** () const



## 7.62 BlackConstantVol Class Reference

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

Inheritance diagram for BlackConstantVol:



### 7.62.1 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](#), and [EquityOption.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 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*

#### Visitability

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

#### Protected Member Functions

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

## 7.63 BlackFormula Class Reference

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

### 7.63.1 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**

## 7.63.2 Member Function Documentation

### 7.63.2.1 [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.

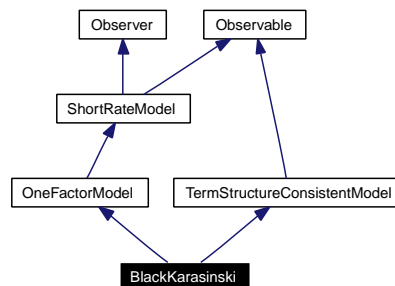
### 7.63.2.2 [Real](#) itmAssetProbability () const

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

## 7.64 BlackKarasinski Class Reference

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

Inheritance diagram for BlackKarasinski:



### 7.64.1 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.*

## 7.65 BlackKarasinski::Dynamics Class Reference

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

### 7.65.1 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.

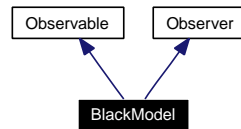
### Public Member Functions

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

## 7.66 BlackModel Class Reference

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

Inheritance diagram for BlackModel:



### 7.66.1 Detailed Description

Black-model for vanilla interest-rate derivatives.

#### Public Member Functions

- **BlackModel** (const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- void [update](#) ()
- [Volatility](#) [volatility](#) () const
- const [Handle](#)< [YieldTermStructure](#) > & [termStructure](#) () const

#### Static Public Member Functions

- static [Real](#) [formula](#) ([Real](#) f, [Real](#) k, [Real](#) v, [Real](#) w)  
*General Black formula.*
- static [Real](#) [itmProbability](#) ([Real](#) f, [Real](#) k, [Real](#) v, [Real](#) w)  
*In-the-money cash probability.*

### 7.66.2 Member Function Documentation

#### 7.66.2.1 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](#).

#### 7.66.2.2 [Real](#) formula ([Real](#) f, [Real](#) k, [Real](#) v, [Real](#) w) [static]

General Black formula.

Returns

$$\text{Black}(f, k, v, w) = fw\Phi(wd_1(f, k, v)) - kw\Phi(wd_2(f, k, v)),$$

where

$$d_1(f, k, v) = \frac{\ln(f/k) + v^2/2}{v}$$

and

$$d_2(f, k, v) = d_1(f, k, v) - v.$$

### 7.66.2.3 **Real** itmProbability (**Real** *f*, **Real** *k*, **Real** *v*, **Real** *w*) [static]

In-the-money cash probability.

Returns

$$P(f, k, v, w) = \Phi(wd_2(f, k, v)),$$

where

$$d_1(f, k, v) = \frac{\ln(f/k) + v^2/2}{v}$$

and

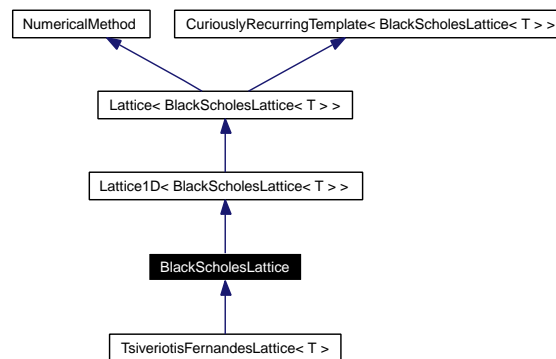
$$d_2(f, k, v) = d_1(f, k, v) - v.$$



## 7.67 BlackScholesLattice Class Template Reference

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

Inheritance diagram for BlackScholesLattice:



### 7.67.1 Detailed Description

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

Simple binomial lattice approximating the Black-Scholes model.

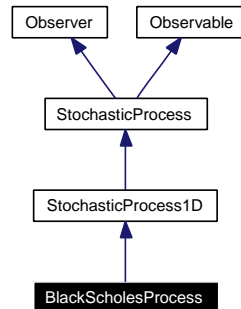
#### Public Member Functions

- **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

## 7.68 BlackScholesProcess Class Reference

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

Inheritance diagram for BlackScholesProcess:



### 7.68.1 Detailed Description

Black-Scholes stochastic process.

This class describes the stochastic process governed by

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

Examples:

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

### Public Member Functions

- **BlackScholesProcess** (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`

## 7.68.2 Member Function Documentation

### 7.68.2.1 [Real](#) drift ([Time](#) *t*, [Real](#) *x*) const [virtual]

#### [Todo](#)

revise extrapolation

Implements [StochasticProcess1D](#).

### 7.68.2.2 [Real](#) diffusion ([Time](#) *t*, [Real](#) *x*) const [virtual]

#### [Todo](#)

revise extrapolation

Implements [StochasticProcess1D](#).

### 7.68.2.3 [Real](#) apply ([Real](#) *x0*, [Real](#) *dx*) const [virtual]

applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented from [StochasticProcess1D](#).

### 7.68.2.4 [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](#).

### 7.68.2.5 `void update ()` [virtual]

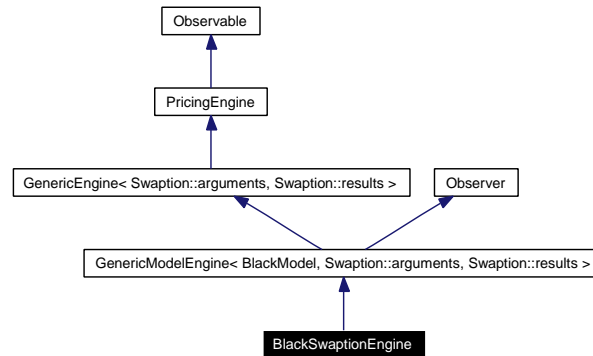
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [StochasticProcess](#).

## 7.69 BlackSwaptionEngine Class Reference

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

Inheritance diagram for BlackSwaptionEngine:



### 7.69.1 Detailed Description

Black-formula swaption engine.

#### Warning

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

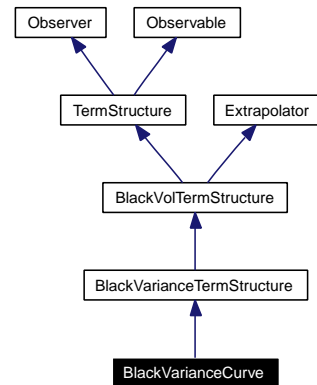
### Public Member Functions

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

## 7.70 BlackVarianceCurve Class Reference

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

Inheritance diagram for BlackVarianceCurve:



### 7.70.1 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 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*

### Modifiers

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

### Visitability

- `virtual void accept (AcyclicVisitor &)`

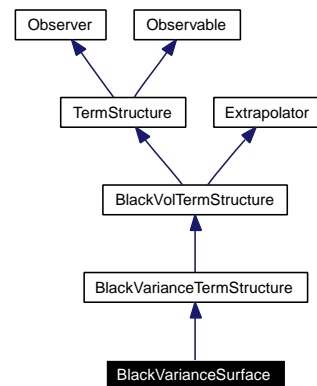
### Protected Member Functions

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

## 7.71 BlackVarianceSurface Class Reference

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

Inheritance diagram for BlackVarianceSurface:



### 7.71.1 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 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*

### Modifiers

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

### Visitability

- `virtual void accept (AcyclicVisitor &)`

### Protected Member Functions

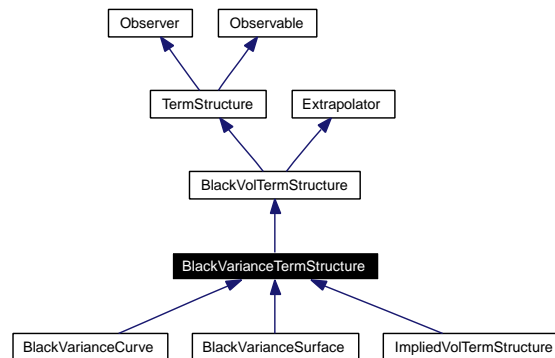
- `virtual Real blackVarianceImpl (Time t, Real strike) const`  
*Black variance calculation.*



## 7.72 BlackVarianceTermStructure Class Reference

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

Inheritance diagram for BlackVarianceTermStructure:



### 7.72.1 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

#### Constructors

See the `TermStructure` documentation for issues regarding constructors.

- [BlackVarianceTermStructure\(\)](#)  
*default constructor*
- [BlackVarianceTermStructure\(const Date &referenceDate\)](#)  
*initialize with a fixed reference date*
- [BlackVarianceTermStructure\(Integer settlementDays, const Calendar &\)](#)  
*calculate the reference date based on the global evaluation date*

#### Visitability

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

### Protected Member Functions

- Volatility [blackVolImpl](#) (Time maturity, Real strike) const

## 7.72.2 Constructor & Destructor Documentation

### 7.72.2.1 [BlackVarianceTermStructure](#) ()

default constructor

#### [Warning](#)

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

## 7.72.3 Member Function Documentation

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

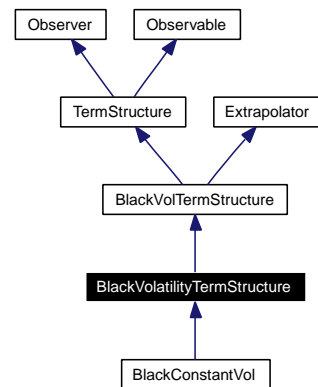
Returns the volatility for the given strike and date calculating it from the variance.

Implements [BlackVolTermStructure](#).

## 7.73 BlackVolatilityTermStructure Class Reference

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

Inheritance diagram for BlackVolatilityTermStructure:



### 7.73.1 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

#### Constructors

See the [TermStructure](#) documentation for issues regarding constructors.

- [BlackVolatilityTermStructure](#) ()  
*default constructor*
- [BlackVolatilityTermStructure](#) (const [Date](#) &referenceDate)  
*initialize with a fixed reference date*
- [BlackVolatilityTermStructure](#) (Integer settlementDays, const [Calendar](#) &)  
*calculate the reference date based on the global evaluation date*

#### Visitability

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

### Protected Member Functions

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

## 7.73.2 Constructor & Destructor Documentation

### 7.73.2.1 [BlackVolatilityTermStructure](#) ()

default constructor

#### [Warning](#)

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

## 7.73.3 Member Function Documentation

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

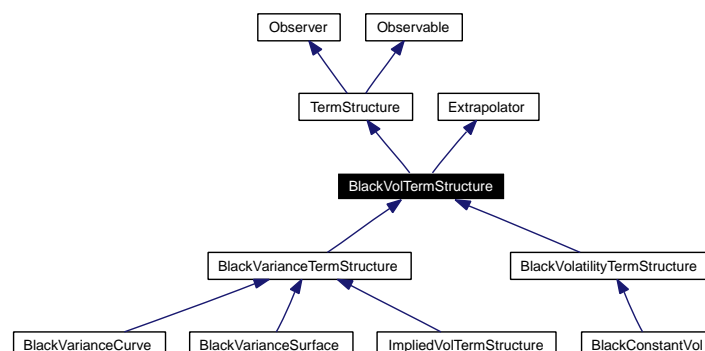
Returns the variance for the given strike and date calculating it from the volatility.

Implements [BlackVolTermStructure](#).

## 7.74 BlackVolTermStructure Class Reference

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

Inheritance diagram for BlackVolTermStructure:



### 7.74.1 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

#### Constructors

See the *TermStructure* documentation for issues regarding constructors.

- [BlackVolTermStructure](#) ()  
*default constructor*
- [BlackVolTermStructure](#) (const [Date](#) &referenceDate)  
*initialize with a fixed reference date*
- [BlackVolTermStructure](#) (Integer settlementDays, const [Calendar](#) &)  
*calculate the reference date based on the global evaluation date*

#### 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 **Date maxDate** () const =0  
*the latest date for which the term structure can return vols*
- **Time maxTime** () const  
*the latest time for which the term structure can return vols*
- 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.*

## 7.74.2 Constructor & Destructor Documentation

### 7.74.2.1 [BlackVolTermStructure](#) ()

default constructor

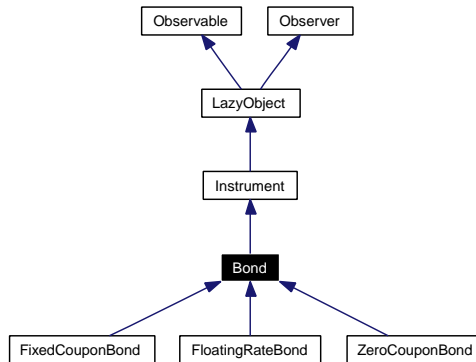
#### **Warning**

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

## 7.75 Bond Class Reference

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

Inheritance diagram for Bond:



### 7.75.1 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
- const std::vector< boost::shared\_ptr< **CashFlow** > > & **cashflows** () const
- const boost::shared\_ptr< **CashFlow** > & **redemption** () const
- const **Calendar** & **calendar** () const
- **BusinessDayConvention** **businessDayConvention** () const
- **BusinessDayConvention** **accrualConvention** () const
- **BusinessDayConvention** **paymentConvention** () 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** (const **DayCounter** &dayCount, const **Calendar** &calendar, **BusinessDayConvention** businessDayConvention, **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\_
- **DayCounter** dayCount\_
- **Date** issueDate\_
- **Date** datedDate\_
- **Date** maturityDate\_
- **Frequency** frequency\_
- **std::vector**< **boost::shared\_ptr**< **CashFlow** > > cashflows\_
- **Handle**< **YieldTermStructure** > discountCurve\_

## 7.75.2 Constructor & Destructor Documentation

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

### Deprecated

use the other constructor

## 7.75.3 Member Function Documentation

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

### Warning

unlike in previous versions, the returned vector now include the redemption as the last cash flow.

7.75.3.2 **BusinessDayConvention** *businessDayConvention* () **const**

### Deprecated

use either *paymentConvention*() or *accrualConvention*()

7.75.3.3 **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.

7.75.3.4 **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.

7.75.3.5 **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.

**7.75.3.6 [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.

**7.75.3.7 [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.

**7.75.3.8 [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.

**7.75.3.9 [Real](#) accruedAmount ([Date](#) d = [Date\(\)](#)) const**

accrued amount at a given date

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

**7.75.3.10 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](#).

## 7.76 BoundaryCondition Class Template Reference

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

### 7.76.1 Detailed Description

`template<class Operator> class QuantLib::BoundaryCondition< Operator >`

Abstract boundary condition class for finite difference problems.

### Public Types

- typedef `Operator` `operator_type`
- typedef `Operator::array_type` `array_type`
- enum [Side](#) { `None`, `Upper`, `Lower` }

### 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

### 7.76.2 Member Enumeration Documentation

#### 7.76.2.1 enum [Side](#)

##### [Todo](#)

Generalize for n-dimensional conditions

### 7.76.3 Member Function Documentation

#### 7.76.3.1 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.

Implemented in [NeumannBC](#), and [DirichletBC](#).

#### 7.76.3.2 virtual void [applyAfterApplying](#) (`array_type` &) const [pure virtual]

This method modifies an array  $u$  so that it satisfies the given condition.

Implemented in [NeumannBC](#), and [DirichletBC](#).

**7.76.3.3 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.

Implemented in [NeumannBC](#), and [DirichletBC](#).

**7.76.3.4 virtual void applyAfterSolving (array\_type &) const** [pure virtual]

This method modifies an array  $u$  so that it satisfies the given condition.

Implemented in [NeumannBC](#), and [DirichletBC](#).

**7.76.3.5 virtual void setTime (Time t)** [pure virtual]

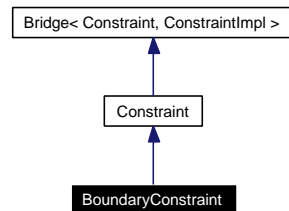
This method sets the current time for time-dependent boundary conditions.

Implemented in [NeumannBC](#), and [DirichletBC](#).

## 7.77 BoundaryConstraint Class Reference

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

Inheritance diagram for BoundaryConstraint:



### 7.77.1 Detailed Description

Constraint imposing all arguments to be in [low,high]

#### Public Member Functions

- **BoundaryConstraint** ([Real](#) low, [Real](#) high)

## 7.78 BoxMullerGaussianRng Class Template Reference

```
#include <ql/RandomNumbers/boxmullergaussianrng.hpp>
```

### 7.78.1 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`

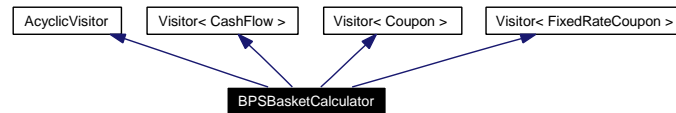
### Public Member Functions

- `BoxMullerGaussianRng` (const RNG &uniformGenerator)
- [sample\\_type next](#) () const  
*returns a sample from a Gaussian distribution*

## 7.79 BPSBasketCalculator Class Reference

```
#include <ql/CashFlows/basispointsensitivity.hpp>
```

Inheritance diagram for BPSBasketCalculator:



### 7.79.1 Detailed Description

#### Bug

this class must still be checked. It is not guaranteed to yield the right results.

### Public Member Functions

- **BPSBasketCalculator** (const [Handle](#)< [YieldTermStructure](#) > &ts, [Integer](#) basis)
- const [TimeBasket](#) & **result** () const

#### Visitor interface

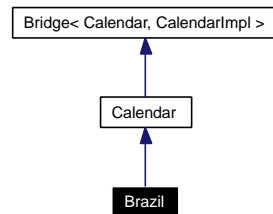
- virtual void **visit** ([Coupon](#) &)
- virtual void **visit** ([FixedRateCoupon](#) &)
- virtual void **visit** ([CashFlow](#) &)



## 7.80 Brazil Class Reference

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

Inheritance diagram for Brazil:



### 7.80.1 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

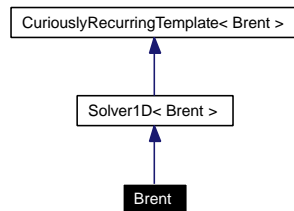
#### Tests

the correctness of the returned results is tested against a list of known holidays.

## 7.81 Brent Class Reference

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

Inheritance diagram for Brent:



### 7.81.1 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`

## 7.82 Bridge Class Template Reference

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

### 7.82.1 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 isNull () const

### Protected Member Functions

- **Bridge** (const boost::shared\_ptr< Impl > &impl=boost::shared\_ptr< Impl >())

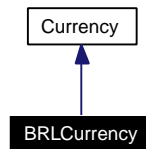
### Protected Attributes

- boost::shared\_ptr< Impl > **impl\_**

## 7.83 BRLCurrency Class Reference

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

Inheritance diagram for BRLCurrency:



### 7.83.1 Detailed Description

Brazilian real.

The ISO three-letter code is BRL; the numeric code is 986. It is divided in 100 centavos.

## 7.84 BrownianBridge Class Template Reference

```
#include <ql/MonteCarlo/brownianbridge.hpp>
```

### 7.84.1 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)

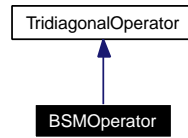
#### inspectors

- const [sample\\_type](#) & **next** () const
- const [sample\\_type](#) & **last** () const
- [Size](#) **size** () const
- const [TimeGrid](#) & **timeGrid** () const

## 7.85 BSMOperator Class Reference

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

Inheritance diagram for BSMOperator:



### 7.85.1 Detailed Description

Black-Scholes-Merton differential operator.

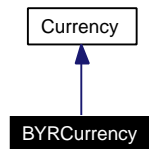
#### Public Member Functions

- **BSMOperator** ([Size](#) size, [Real](#) dx, [Rate](#) r, [Rate](#) q, [Volatility](#) sigma)
- **BSMOperator** (const [Array](#) &grid, const boost::shared\_ptr< [BlackScholesProcess](#) > &, [Time](#) residualTime)

## 7.86 BYRCurrency Class Reference

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

Inheritance diagram for BYRCurrency:



### 7.86.1 Detailed Description

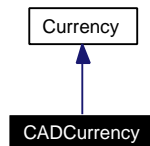
Belarussian ruble.

The ISO three-letter code is BYR; the numeric code is 974. It has no subdivisions.

## 7.87 CADCurrency Class Reference

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

Inheritance diagram for CADCurrency:



### 7.87.1 Detailed Description

Canadian dollar.

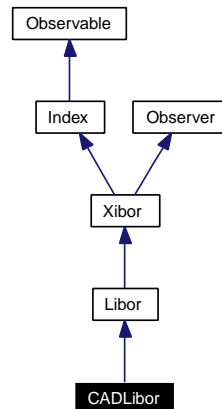
The ISO three-letter code is CAD; the numeric code is 124. It is divided into 100 cents.



## 7.88 CADLibor Class Reference

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

Inheritance diagram for CADLibor:



### 7.88.1 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.

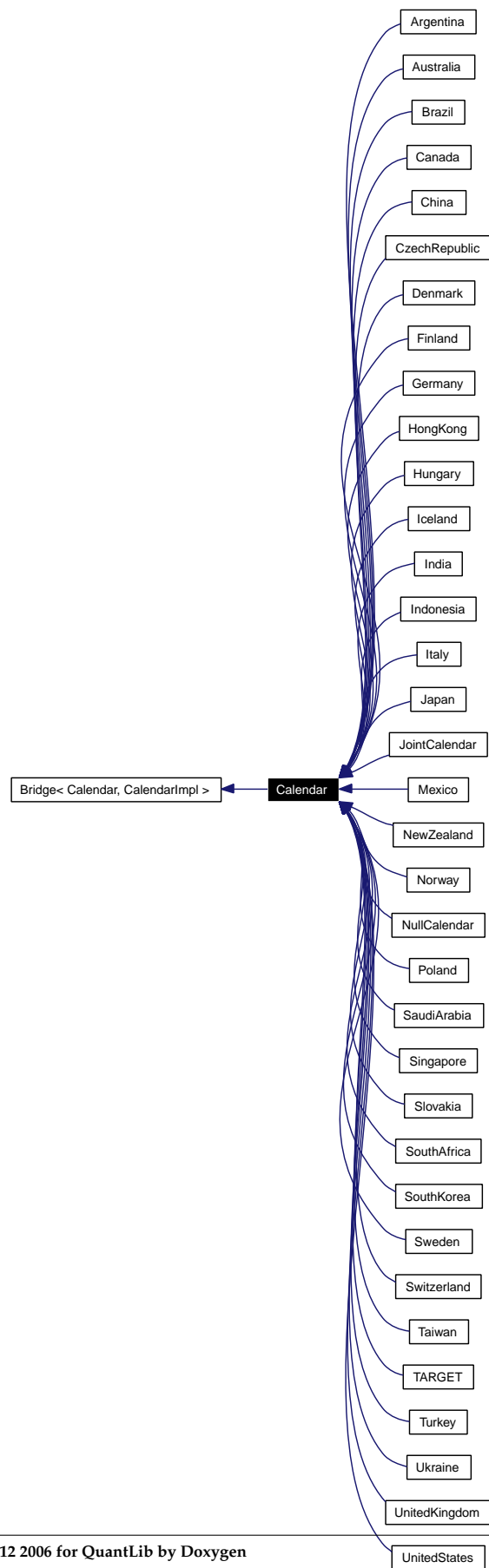
### Public Member Functions

- **CADLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.89 Calendar Class Reference

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

Inheritance diagram for Calendar:



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

### Public Member Functions

- [Calendar](#) ()

#### 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 isEndOfMonth (const Date &d) const`
- `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) const`
- `Date advance (const Date &date, const Period &period, BusinessDayConvention convention=Following) const`

### 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 &)`

## Classes

- class [OrthodoxImpl](#)  
*partial calendar implementation*
- class [WesternImpl](#)  
*partial calendar implementation*

## 7.89.2 Constructor & Destructor Documentation

### 7.89.2.1 [Calendar](#) ()

This default constructor returns a calendar with a null implementation, which is therefore unusable except as a placeholder.

### 7.89.2.2 [Calendar](#) (const boost::shared\_ptr< [CalendarImpl](#) > & *impl*) [protected]

This protected constructor will only be invoked by derived classes which define a given [Calendar](#) implementation

## 7.89.3 Member Function Documentation

### 7.89.3.1 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.

### 7.89.3.2 bool isBusinessDay (const [Date](#) & *d*) const

Returns true iff the date is a business day for the given market.

### 7.89.3.3 bool isHoliday (const [Date](#) & *d*) const

Returns true iff the date is a holiday for the given market.

### 7.89.3.4 bool isEndOfMonth (const [Date](#) & *d*) const

Returns true iff the date is last business day for the month in given market.

### 7.89.3.5 void addHoliday (const [Date](#) &)

Adds a date to the set of holidays for the given calendar.

#### 7.89.3.6 void removeHoliday (const [Date](#) &)

Removes a date from the set of holidays for the given calendar.

#### 7.89.3.7 [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](#).

#### 7.89.3.8 [Date](#) advance (const [Date](#) &, [Integer](#) *n*, [TimeUnit](#) *unit*, [BusinessDayConvention](#) *convention* = Following) const

Advances the given date of the given number of business days and returns the result.

Note:

The input date is not modified.

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

#### 7.89.3.9 [Date](#) advance (const [Date](#) & *date*, const [Period](#) & *period*, [BusinessDayConvention](#) *convention* = Following) const

Advances the given date as specified by the given period and returns the result.

Note:

The input date is not modified.

### 7.89.4 Friends And Related Function Documentation

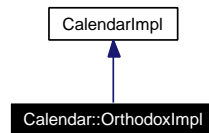
#### 7.89.4.1 bool operator== (const [Calendar](#) &, const [Calendar](#) &) [related]

Returns true iff the two calendars belong to the same derived class.

## 7.90 Calendar::OrthodoxImpl Class Reference

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

Inheritance diagram for Calendar::OrthodoxImpl:



### 7.90.1 Detailed Description

partial calendar implementation

This class provides the means of determining the Easter Monday for a given year.

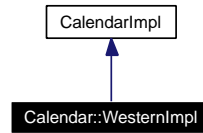
#### Static Protected Member Functions

- static [Day](#) [easterMonday](#) ([Year](#) y)  
*expressed relative to first day of year*

## 7.91 Calendar::WesternImpl Class Reference

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

Inheritance diagram for Calendar::WesternImpl:



### 7.91.1 Detailed Description

partial calendar implementation

This class provides the means of determining the Easter Monday for a given year.

#### Static Protected Member Functions

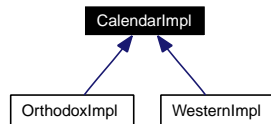
- static [Day](#) [easterMonday](#) ([Year](#) y)  
*expressed relative to first day of year*



## 7.92 CalendarImpl Class Reference

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

Inheritance diagram for CalendarImpl:



### 7.92.1 Detailed Description

abstract base class for calendar implementations

#### Public Member Functions

- virtual std::string **name** () const =0
- virtual bool **isBusinessDay** (const [Date](#) &) const =0

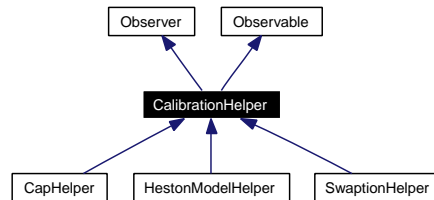
#### Public Attributes

- std::set< [Date](#) > **addedHolidays**
- std::set< [Date](#) > **removedHolidays**

## 7.93 CalibrationHelper Class Reference

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

Inheritance diagram for CalibrationHelper:



### 7.93.1 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< [BlackModel](#) > [blackModel\\_](#)
- boost::shared\_ptr< [PricingEngine](#) > [engine\\_](#)

## 7.93.2 Member Function Documentation

### 7.93.2.1 void update () [virtual]

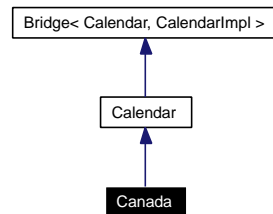
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.94 Canada Class Reference

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

Inheritance diagram for Canada:



### 7.94.1 Detailed Description

Canadian calendar.

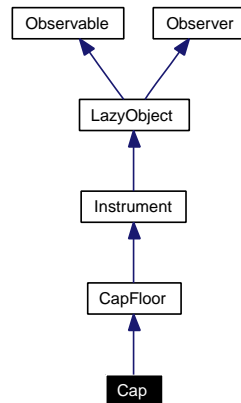
Holidays:

- 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)

## 7.95 Cap Class Reference

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

Inheritance diagram for Cap:



### 7.95.1 Detailed Description

Concrete cap class.

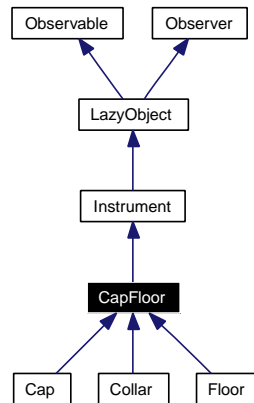
#### Public Member Functions

- `Cap` (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &floatingLeg, const std::vector< [Rate](#) > &exerciseRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared\_ptr< [PricingEngine](#) > &engine)

## 7.96 CapFloor Class Reference

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

Inheritance diagram for CapFloor:



### 7.96.1 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

### Classes

- class [arguments](#)  
*Arguments for cap/floor calculation*
- class [results](#)  
*Results from cap/floor calculation*

## 7.96.2 Member Function Documentation

### 7.96.2.1 void `setupArguments (Arguments *)` const [virtual]

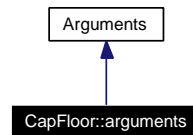
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

## 7.97 CapFloor::arguments Class Reference

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

Inheritance diagram for CapFloor::arguments:



### 7.97.1 Detailed Description

Arguments for cap/floor calculation

#### Public Member Functions

- void **validate** () const

#### Public Attributes

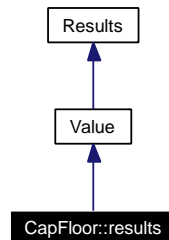
- CapFloor::Type **type**
- std::vector< [Time](#) > **startTimes**
- std::vector< [Time](#) > **fixingTimes**
- std::vector< [Time](#) > **endTimes**
- std::vector< [Time](#) > **accrualTimes**
- std::vector< [Rate](#) > **capRates**
- std::vector< [Rate](#) > **floorRates**
- std::vector< [Rate](#) > **forwards**
- std::vector< [Real](#) > **nominals**



## 7.98 CapFloor::results Class Reference

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

Inheritance diagram for CapFloor::results:



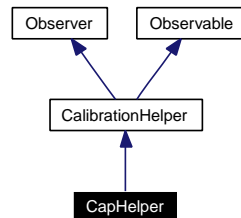
### 7.98.1 Detailed Description

Results from cap/floor calculation

## 7.99 CapHelper Class Reference

```
#include <ql/ShortRateModels/CalibrationHelpers/caphelper.hpp>
```

Inheritance diagram for CapHelper:



### 7.99.1 Detailed Description

calibration helper for ATM cap

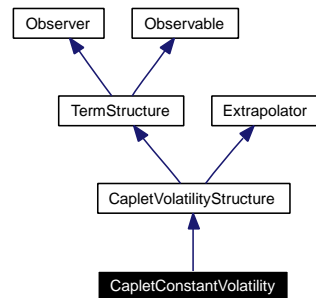
#### Public Member Functions

- **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.*

## 7.100 CapletConstantVolatility Class Reference

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

Inheritance diagram for CapletConstantVolatility:



### 7.100.1 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** ([Integer](#) settlementDays, const [Calendar](#) &, [Volatility](#) volatility, const [DayCounter](#) &dayCounter)
- **CapletConstantVolatility** ([Integer](#) settlementDays, const [Calendar](#) &, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- [Date](#) **maxDate** () const  
*the latest date for which the term structure can return vols*
- [Time](#) **maxTime** () const  
*the latest time for which the term structure can return vols*
- [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

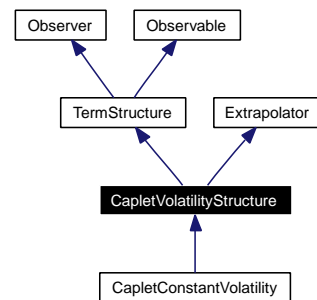
### CapletVolatilityStructure interface

- [Volatility](#) [volatilityImpl](#) ([Time](#) t, [Rate](#)) const  
*implements the actual volatility calculation in derived classes*

## 7.101 CapletVolatilityStructure Class Reference

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

Inheritance diagram for CapletVolatilityStructure:



### 7.101.1 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

#### Constructors

See the *TermStructure* documentation for issues regarding constructors.

- [CapletVolatilityStructure](#) ()  
*default constructor*
- [CapletVolatilityStructure](#) (const [Date](#) &referenceDate)  
*initialize with a fixed reference date*
- [CapletVolatilityStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)  
*calculate the reference date based on the global evaluation date*

#### Volatility

- [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*

#### Limits

- virtual [Date maxDate](#) () const =0  
*the latest date for which the term structure can return vols*

- virtual [Time](#) `maxTime` () const  
*the latest time for which the term structure can return vols*
- 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*

## 7.101.2 Constructor & Destructor Documentation

### 7.101.2.1 [CapletVolatilityStructure](#) ()

default constructor

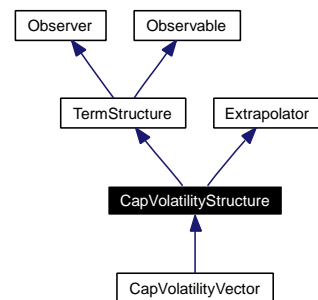
#### [Warning](#)

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

## 7.102 CapVolatilityStructure Class Reference

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

Inheritance diagram for CapVolatilityStructure:



### 7.102.1 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

#### Constructors

See the *TermStructure* documentation for issues regarding constructors.

- [CapVolatilityStructure](#) ()  
*default constructor*
- [CapVolatilityStructure](#) (const [Date](#) &referenceDate)  
*initialize with a fixed reference date*
- [CapVolatilityStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)  
*calculate the reference date based on the global evaluation date*

#### 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 [Date maxDate](#) () const =0

*the latest date for which the term structure can return vols*

- virtual [Time](#) [maxTime](#) () const  
*the latest time for which the term structure can return vols*
- 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*

## 7.102.2 Constructor & Destructor Documentation

### 7.102.2.1 [CapVolatilityStructure](#) ()

default constructor

#### [Warning](#)

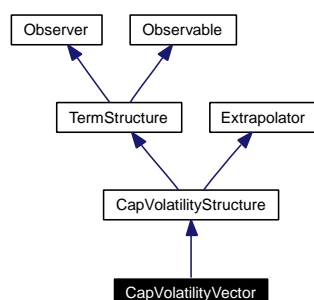
term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.



## 7.103 CapVolatilityVector Class Reference

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

Inheritance diagram for CapVolatilityVector:



### 7.103.1 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 term structure can return vols*
- [Time](#) maxTime () const  
*the latest time 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*
- void [update](#) ()

## 7.103.2 Member Function Documentation

### 7.103.2.1 `void update ()` [virtual]

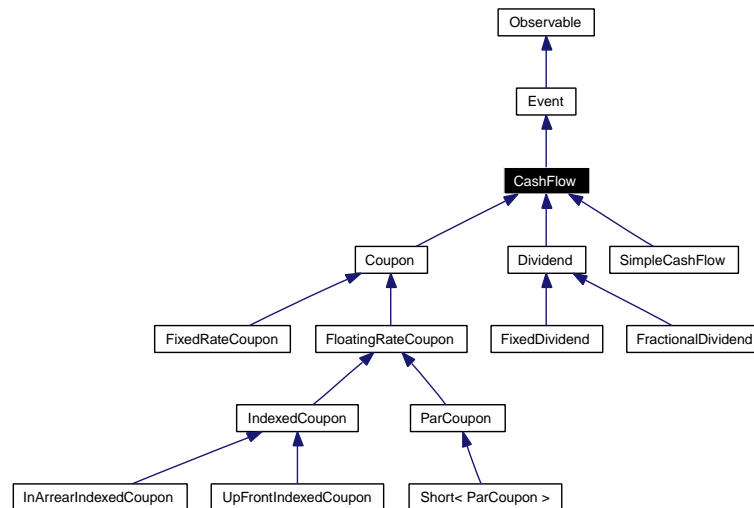
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

## 7.104 CashFlow Class Reference

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

Inheritance diagram for CashFlow:



### 7.104.1 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

#### Visitability

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

### 7.104.2 Member Function Documentation

#### 7.104.2.1 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](#), [IndexedCoupon](#), [ParCoupon](#), [Short< ParCoupon >](#), and [SimpleCashFlow](#).

7.104.2.2 **virtual [Date](#) date () const** [pure virtual]

**Note:**

This is inherited from the event class

Implements [Event](#).

Implemented in [Coupon](#), [Dividend](#), and [SimpleCashFlow](#).

## 7.105 Cashflows Class Reference

```
#include <ql/CashFlows/analysis.hpp>
```

### 7.105.1 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.*

### 7.105.2 Member Function Documentation

- 7.105.2.1 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.

**7.105.2.2** 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.

**7.105.2.3** 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.

**7.105.2.4** 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.

**7.105.2.5** 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.

**7.105.2.6** 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.

**7.105.2.7 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

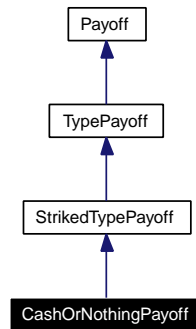
$$C = \frac{1}{P} \frac{\partial^2 P}{\partial y^2}$$

where  $P$  is the present value of the cash flows according to the given IRR  $y$ .

## 7.106 CashOrNothingPayoff Class Reference

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

Inheritance diagram for CashOrNothingPayoff:



### 7.106.1 Detailed Description

Binary cash-or-nothing payoff.

#### Public Member Functions

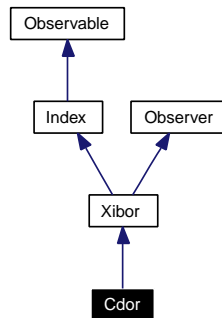
- **CashOrNothingPayoff** (Option::Type type, [Real](#) strike, [Real](#) cashPayoff)
- [Real](#) **operator()** ([Real](#) price) const
- [Real](#) **cashPayoff** () const
- virtual void **accept** ([AcyclicVisitor](#) &)



## 7.107 Cdor Class Reference

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

Inheritance diagram for Cdor:



### 7.107.1 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.

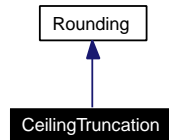
### Public Member Functions

- **Cdor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.108 CeilingTruncation Class Reference

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

Inheritance diagram for CeilingTruncation:



### 7.108.1 Detailed Description

Ceiling truncation.

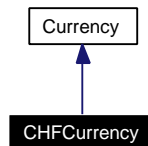
#### Public Member Functions

- **CeilingTruncation** ([Integer](#) precision, [Integer](#) digit=5)

## 7.109 CHFCurrency Class Reference

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

Inheritance diagram for CHFCurrency:



### 7.109.1 Detailed Description

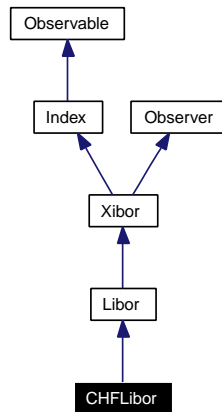
Swiss franc.

The ISO three-letter code is CHF; the numeric code is 756. It is divided into 100 cents.

## 7.110 CHFLibor Class Reference

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

Inheritance diagram for CHFLibor:



### 7.110.1 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.

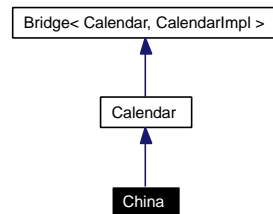
### Public Member Functions

- **CHFLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.111 China Class Reference

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

Inheritance diagram for China:



### 7.111.1 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

## 7.112 CLGaussianRng Class Template Reference

```
#include <ql/RandomNumbers/centrallimitgaussianrng.hpp>
```

### 7.112.1 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**

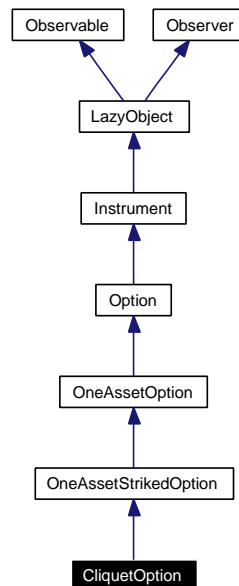
### Public Member Functions

- **CLGaussianRng** (const RNG &uniformGenerator)
- [sample\\_type next](#) () const  
*returns a sample from a Gaussian distribution*

## 7.113 CliquetOption Class Reference

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

Inheritance diagram for CliquetOption:



### 7.113.1 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.*

## 7.113.2 Member Function Documentation

### 7.113.2.1 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](#).



## 7.114 CliquetOption::arguments Class Reference

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

### 7.114.1 Detailed Description

Arguments for cliquet option calculation

#### Public Member Functions

- void **validate** () const

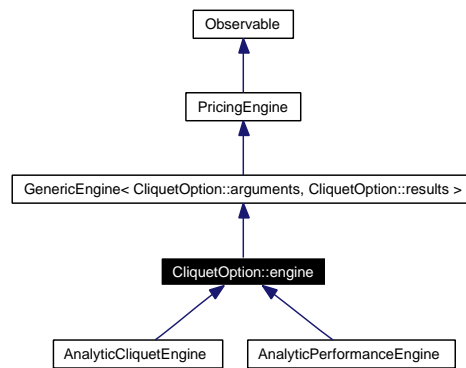
#### Public Attributes

- [Real](#) **accruedCoupon**
- [Real](#) **lastFixing**
- [Real](#) **localCap**
- [Real](#) **localFloor**
- [Real](#) **globalCap**
- [Real](#) **globalFloor**
- std::vector< [Date](#) > **resetDates**

## 7.115 CliquetOption::engine Class Reference

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

Inheritance diagram for CliquetOption::engine:



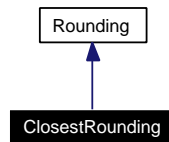
### 7.115.1 Detailed Description

Cliquet engine base class.

## 7.116 ClosestRounding Class Reference

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

Inheritance diagram for ClosestRounding:



### 7.116.1 Detailed Description

Closest rounding.

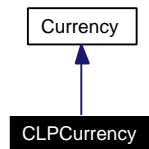
#### Public Member Functions

- `ClosestRounding` (`Integer` precision, `Integer` digit=5)

## 7.117 CLPCurrency Class Reference

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

Inheritance diagram for CLPCurrency:



### 7.117.1 Detailed Description

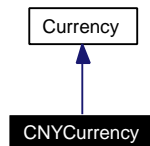
Chilean peso.

The ISO three-letter code is CLP; the numeric code is 152. It is divided in 100 centavos.

## 7.118 CNYCurrency Class Reference

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

Inheritance diagram for CNYCurrency:



### 7.118.1 Detailed Description

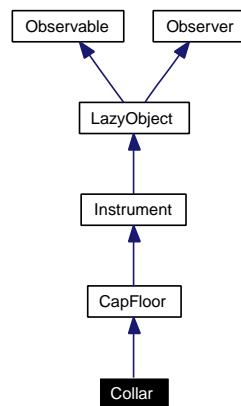
Chinese yuan.

The ISO three-letter code is CNY; the numeric code is 156. It is divided in 100 fen.

## 7.119 Collar Class Reference

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

Inheritance diagram for Collar:



### 7.119.1 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)

## 7.120 Composite Class Template Reference

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

### 7.120.1 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)

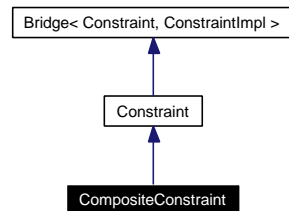
### Protected Attributes

- std::list< boost::shared\_ptr< T > > **components\_**

## 7.121 CompositeConstraint Class Reference

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

Inheritance diagram for CompositeConstraint:



### 7.121.1 Detailed Description

Constraint enforcing both given sub-constraints

#### Public Member Functions

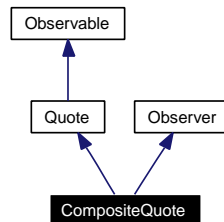
- **CompositeConstraint** (const [Constraint](#) &c1, const [Constraint](#) &c2)



## 7.122 CompositeQuote Class Template Reference

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

Inheritance diagram for CompositeQuote:



### 7.122.1 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](#) ()

### 7.122.2 Member Function Documentation

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

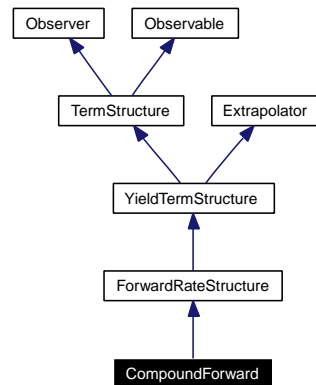
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.123 CompoundForward Class Reference

```
#include <ql/TermStructures/compoundforward.hpp>
```

Inheritance diagram for CompoundForward:



### 7.123.1 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 rates*
- [Time](#) **maxTime** () const

*the latest time for which the curve can return rates*

- `const std::vector< Time > & times () const`
- `const std::vector< Date > & dates () const`
- `const std::vector< 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`

## 7.123.2 Member Function Documentation

### 7.123.2.1 [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](#).

### 7.123.2.2 [DiscountFactor](#) discountImpl ([Time](#)) const [protected, virtual]

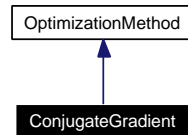
Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Reimplemented from [ForwardRateStructure](#).

## 7.124 ConjugateGradient Class Reference

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

Inheritance diagram for ConjugateGradient:



### 7.124.1 Detailed Description

Multi-dimensional Conjugate Gradient class.

User has to provide line-search method and optimization end criteria

search direction  $d_i = -f'(x_i) + c_i * d_{i-1}$  where  $c_i = \|f'(x_i)\|^2 / \|f'(x_{i-1})\|^2$  and  $d_1 = -f'(x_1)$

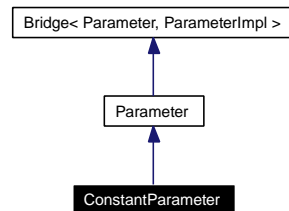
### Public Member Functions

- [ConjugateGradient](#) ()  
*default constructor*
- **ConjugateGradient** (const boost::shared\_ptr< [LineSearch](#) > &lineSearch)
- virtual [~ConjugateGradient](#) ()  
*destructor*
- virtual void [minimize](#) (const [Problem](#) &P) const  
*minimize the optimization problem P*

## 7.125 ConstantParameter Class Reference

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

Inheritance diagram for ConstantParameter:



### 7.125.1 Detailed Description

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

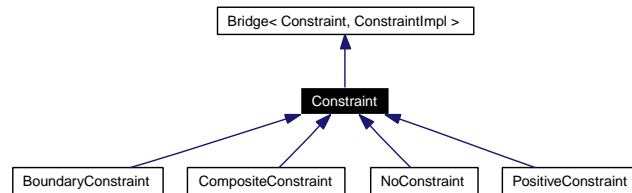
#### Public Member Functions

- **ConstantParameter** (const [Constraint](#) &constraint)
- **ConstantParameter** ([Real](#) value, const [Constraint](#) &constraint)

## 7.126 Constraint Class Reference

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

Inheritance diagram for Constraint:



### 7.126.1 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](#) >())

## 7.127 ConstraintImpl Class Reference

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

### 7.127.1 Detailed Description

Base class for constraint implementations.

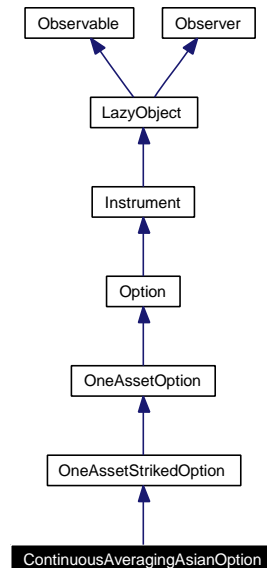
#### Public Member Functions

- virtual bool [test](#) (const [Array](#) &params) const =0  
*Tests if params satisfy the constraint.*

## 7.128 ContinuousAveragingAsianOption Class Reference

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

Inheritance diagram for ContinuousAveragingAsianOption:



### 7.128.1 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.*

## 7.128.2 Member Function Documentation

### 7.128.2.1 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](#).

## 7.129 ContinuousAveragingAsianOption::arguments Class Reference

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

### 7.129.1 Detailed Description

Extra arguments for single-asset continuous-average Asian option.

#### Public Member Functions

- void **validate** () const

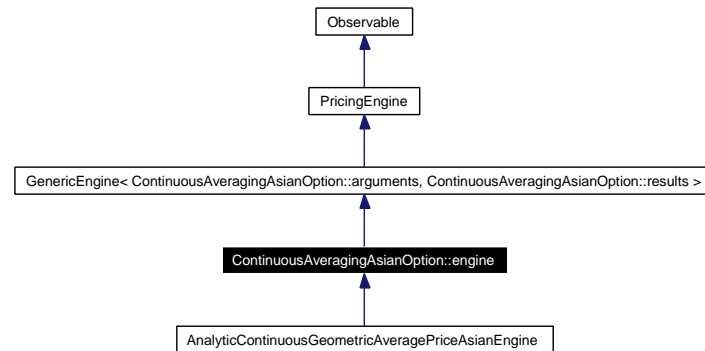
#### Public Attributes

- Average::Type **averageType**

## 7.130 ContinuousAveragingAsianOption::engine Class Reference

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

Inheritance diagram for ContinuousAveragingAsianOption::engine:



### 7.130.1 Detailed Description

Continuous-averaging Asian engine base class.

## 7.131 ConvergenceStatistics Class Template Reference

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

### 7.131.1 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 Member Functions

- **ConvergenceStatistics** (const U &rule=U())
- void **add** (Real 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, Real > > & **convergenceTable** () const

## 7.132 ConvertibleBond::option::arguments Class Reference

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

### 7.132.1 Detailed Description

Arguments for Convertible [Bond](#) calculation

#### Public Member Functions

- void **validate** () const

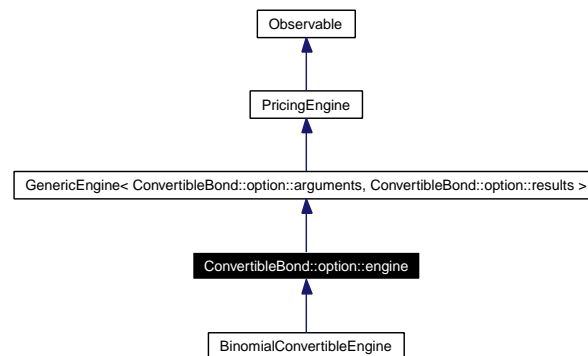
#### Public Attributes

- [Real](#) **conversionRatio**
- [Handle](#)< [Quote](#) > **creditSpread**
- DividendSchedule **dividends**
- std::vector< [Time](#) > **callabilityTimes**
- std::vector< Callability::Type > **callabilityTypes**
- std::vector< [Real](#) > **callabilityPrices**
- std::vector< [Time](#) > **couponTimes**
- std::vector< [Real](#) > **couponAmounts**
- [DayCounter](#) **dayCounter**
- [Date](#) **issueDate**
- [Date](#) **settlementDate**
- [Integer](#) **settlementDays**
- [Real](#) **redemption**

## 7.133 ConvertibleBond::option::engine Class Reference

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

Inheritance diagram for ConvertibleBond::option::engine:



### 7.133.1 Detailed Description

convertible bond engine base class

## 7.134 ConvertibleFixedCouponBond Class Reference

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

### 7.134.1 Detailed Description

convertible fixed-coupon bond

#### Warning

At this time, discrete dividends are not managed.

#### 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)

## 7.135 ConvertibleFloatingRateBond Class Reference

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

### 7.135.1 Detailed Description

convertible floating-rate bond

#### Warning

At this time, discrete dividends are not managed.

#### 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)



## 7.136 ConvertibleZeroCouponBond Class Reference

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

### 7.136.1 Detailed Description

convertible zero-coupon bond

#### Warning

At this time, discrete dividends are not managed.

#### 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.

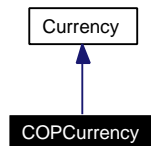
### Public Member Functions

- **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)

## 7.137 COPCurrency Class Reference

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

Inheritance diagram for COPCurrency:



### 7.137.1 Detailed Description

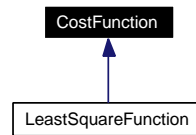
Colombian peso.

The ISO three-letter code is COP; the numeric code is 170. It is divided in 100 centavos.

## 7.138 CostFunction Class Reference

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

Inheritance diagram for CostFunction:



### 7.138.1 Detailed Description

Cost function abstract class for optimization problem.

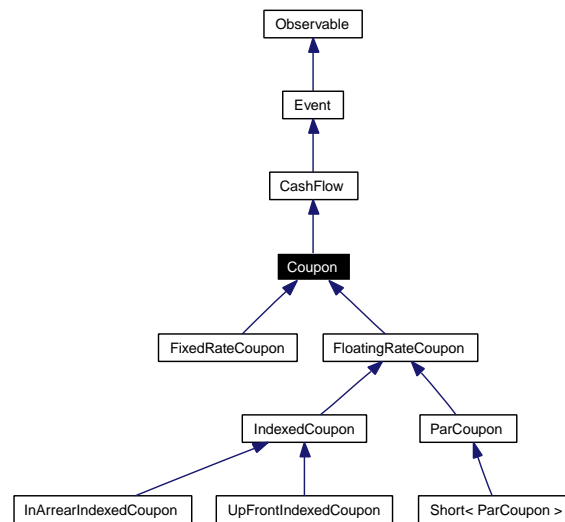
#### Public Member Functions

- 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 .:*

## 7.139 Coupon Class Reference

```
#include <ql/CashFlows/coupon.hpp>
```

Inheritance diagram for Coupon:



### 7.139.1 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

#### 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_`

## 7.139.2 Constructor & Destructor Documentation

- 7.139.2.1 [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.

## 7.139.3 Member Function Documentation

- 7.139.3.1 [Date](#) `date` () const [virtual]

### Note:

This is inherited from the event class

Implements [CashFlow](#).

## 7.140 CovarianceDecomposition Class Reference

```
#include <ql/MonteCarlo/getcovariance.hpp>
```

### 7.140.1 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

### 7.140.2 Constructor & Destructor Documentation

**7.140.2.1** [CovarianceDecomposition](#) (const [Matrix](#) & *covarianceMatrix*, [Real](#) *tolerance* = 1.0e-12)

#### Precondition:

*covarianceMatrix* must be symmetric

### 7.140.3 Member Function Documentation

**7.140.3.1** const [Array](#)& [variances](#) () const

returns the variances [Array](#)

**7.140.3.2** const [Array](#)& [standardDeviations](#) () const

returns the standard deviations [Array](#)

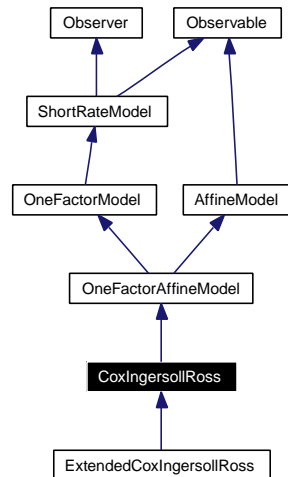
**7.140.3.3** const [Matrix](#)& [correlationMatrix](#) () const

returns the correlation matrix

## 7.141 CoxIngersollRoss Class Reference

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

Inheritance diagram for CoxIngersollRoss:



### 7.141.1 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

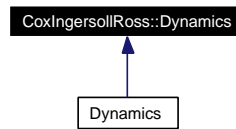
- class [Dynamics](#)  
*Dynamics of the short-rate under the Cox-Ingersoll-Ross model*



## 7.142 CoxIngersollRoss::Dynamics Class Reference

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

Inheritance diagram for CoxIngersollRoss::Dynamics:



### 7.142.1 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

$$dy_t = \left[ \left( \frac{k\theta}{2} + \frac{\sigma^2}{8} \right) \frac{1}{y_t} - \frac{k}{2} y_t \right] dt + \frac{\sigma}{2} dW_t$$

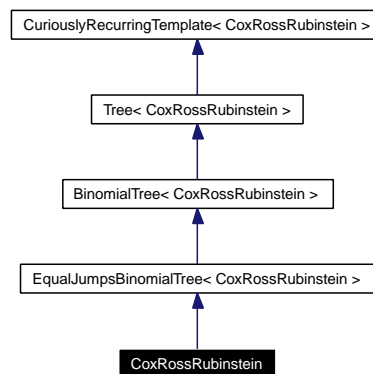
### Public Member Functions

- **Dynamics** ([Real](#) theta, [Real](#) k, [Real](#) sigma, [Real](#) x0)
- virtual [Real](#) **variable** ([Time](#), [Rate](#) r) const
- virtual [Real](#) **shortRate** ([Time](#), [Real](#) y) const

## 7.143 CoxRossRubinstein Class Reference

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

Inheritance diagram for CoxRossRubinstein:



### 7.143.1 Detailed Description

Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.

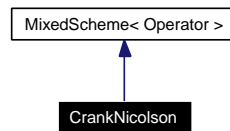
#### Public Member Functions

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

## 7.144 CrankNicolson Class Template Reference

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

Inheritance diagram for CrankNicolson:



### 7.144.1 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)

## 7.145 Cubic Class Reference

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

### 7.145.1 Detailed Description

cubic-spline interpolation factory and traits

#### Public Types

- enum { **global** = 1 }

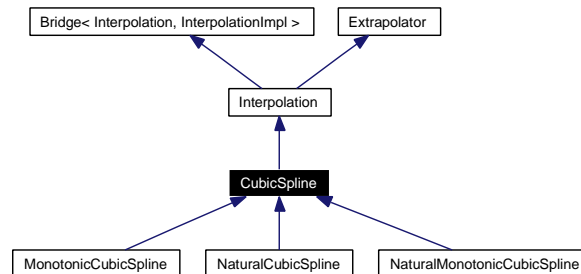
#### Public Member Functions

- **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

## 7.146 CubicSpline Class Reference

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

Inheritance diagram for CubicSpline:



### 7.146.1 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

## 7.146.2 Member Enumeration Documentation

### 7.146.2.1 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

## 7.146.3 Constructor & Destructor Documentation

7.146.3.1 [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.

## 7.147 CumulativeBinomialDistribution Class Reference

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

### 7.147.1 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



## 7.148 CumulativeNormalDistribution Class Reference

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

### 7.148.1 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

## 7.149 CumulativePoissonDistribution Class Reference

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

### 7.149.1 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.

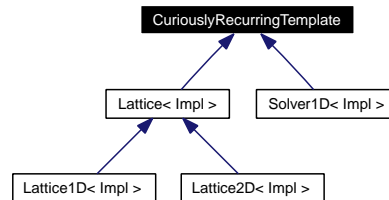
### Public Member Functions

- **CumulativePoissonDistribution** ([Real](#) mu)
- **Real operator()** (BigNatural k) const

## 7.150 CuriouslyRecurringTemplate Class Template Reference

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

Inheritance diagram for CuriouslyRecurringTemplate:



### 7.150.1 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.

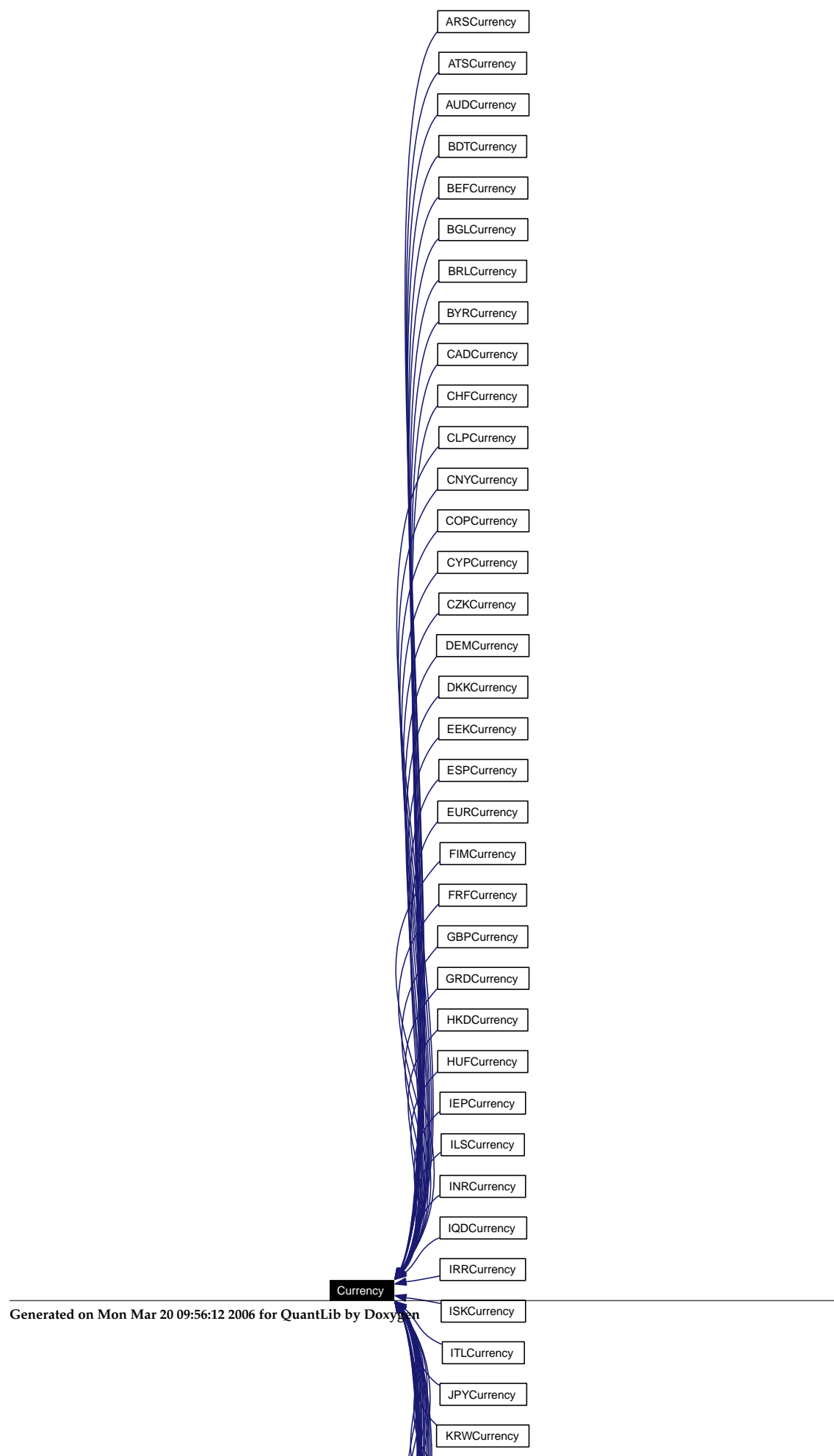
### Protected Member Functions

- Impl & impl ()
- const Impl & impl () const

## 7.151 Currency Class Reference

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

Inheritance diagram for Currency:



### 7.151.1 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](#) &)

## 7.151.2 Constructor & Destructor Documentation

### 7.151.2.1 [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.

## 7.151.3 Member Function Documentation

### 7.151.3.1 `std::string format () const`

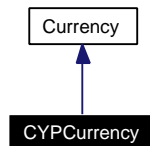
output format

The format will be fed three positional parameters, namely, value, code, and symbol, in this order.

## 7.152 CYPCurrency Class Reference

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

Inheritance diagram for CYPCurrency:



### 7.152.1 Detailed Description

Cyprus pound.

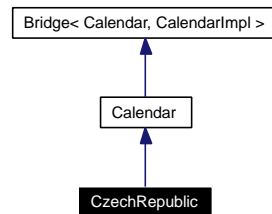
The ISO three-letter code is CYP; the numeric code is 196. It is divided in 100 cents.



## 7.153 CzechRepublic Class Reference

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

Inheritance diagram for CzechRepublic:



### 7.153.1 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)

## 7.153.2 Member Enumeration Documentation

### 7.153.2.1 enum [Market](#)

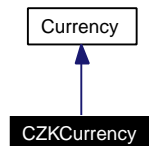
Enumerator:

*PSE* Prague stock exchange.

## 7.154 CZKCurrency Class Reference

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

Inheritance diagram for CZKCurrency:



### 7.154.1 Detailed Description

Czech koruna.

The ISO three-letter code is CZK; the numeric code is 203. It is divided in 100 haleru.

## 7.155 Date Class Reference

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

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

## Public Member Functions

### constructors

- [Date](#) ()  
*Default constructor returning a null date.*
- [Date](#) ([BigInteger](#) serialNumber)  
*Constructor taking a serial number as given by Applix or Excel.*
- [Date](#) ([Day](#) d, [Month](#) m, [Year](#) y)  
*More traditional constructor.*

### inspectors

- [Weekday](#) [weekday](#) () const
- [Day](#) [dayOfMonth](#) () const
- [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)  
*whether or not the given date is an IMM date*
- static [Date](#) [nextIMMdate](#) (const [Date](#) &d)  
*next IMM date following (or equal to) the given 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](#) &)

## 7.155.2 Member Function Documentation

### 7.155.2.1 static [Date](#) [nextWeekday](#) (const [Date](#) & d, [Weekday](#)) [static]

next given weekday following or equal to the given date

E.g., the Friday following January 15th, 20 (a Tuesday) was January 18th, 2002.

see <http://www.cpearson.com/excel/DateTimeWS.htm>

### 7.155.2.2 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>

### 7.155.2.3 static [Date](#) [nextIMMdate](#) (const [Date](#) & d) [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.

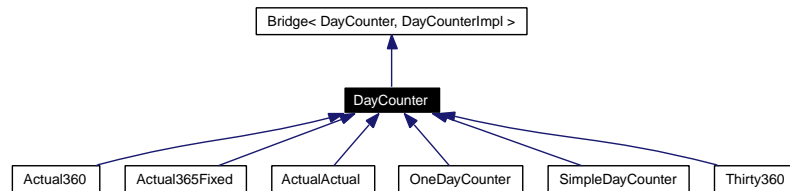
**Warning**

The result date is following or equal to the original date.

## 7.156 DayCounter Class Reference

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

Inheritance diagram for DayCounter:



### 7.156.1 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](#), 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 &)`

## 7.156.2 Constructor & Destructor Documentation

### 7.156.2.1 DayCounter ()

This default constructor returns a day counter with a null implementation, which is therefore unusable except as a placeholder.

### 7.156.2.2 DayCounter (const boost::shared\_ptr< DayCounterImpl > & impl) [protected]

This protected constructor will only be invoked by derived classes which define a given DayCounter implementation

## 7.156.3 Member Function Documentation

### 7.156.3.1 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.

## 7.156.4 Friends And Related Function Documentation

### 7.156.4.1 bool operator==(const DayCounter &, const DayCounter &) [related]

Returns true iff the two day counters belong to the same derived class.

## 7.157 DayCounterImpl Class Reference

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

### 7.157.1 Detailed Description

abstract base class for day counter implementations

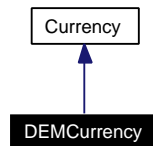
#### Public Member Functions

- 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`

## 7.158 DEMCurrency Class Reference

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

Inheritance diagram for DEMCurrency:



### 7.158.1 Detailed Description

Deutsche mark.

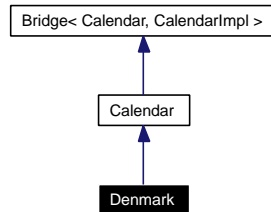
The ISO three-letter code was DEM; the numeric code was 276. It was divided into 100 pfennig.

Obsoleted by the Euro since 1999.

## 7.159 Denmark Class Reference

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

Inheritance diagram for Denmark:



### 7.159.1 Detailed Description

Danish calendar.

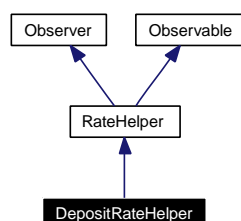
Holidays:

- 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

## 7.160 DepositRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for DepositRateHelper:



### 7.160.1 Detailed Description

Deposit rate helper.

#### Warning

This class assumes that the reference date does not change between calls of `setTermStructure()`.

#### Examples:

`swapvaluation.cpp`.

### Public Member Functions

- **DepositRateHelper** (const [Handle](#)< [Quote](#) > &rate, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **DepositRateHelper** ([Rate](#) rate, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **Real impliedQuote** () const
- **DiscountFactor discountGuess** () const
- void **setTermStructure** ([YieldTermStructure](#) \*)  
*sets the term structure to be used for pricing*
- **Date latestDate** () const  
*latest relevant date*

### 7.160.2 Member Function Documentation

#### 7.160.2.1 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](#).

#### 7.160.2.2 [Date](#) latestDate () const [virtual]

latest relevant date

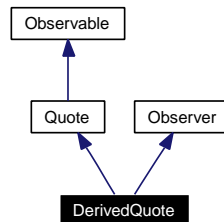
The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

## 7.161 DerivedQuote Class Template Reference

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

Inheritance diagram for DerivedQuote:



### 7.161.1 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](#) ()

### 7.161.2 Member Function Documentation

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

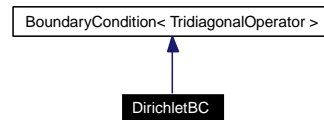
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.162 DirichletBC Class Reference

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

Inheritance diagram for DirichletBC:



### 7.162.1 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](#))

### 7.162.2 Member Function Documentation

#### 7.162.2.1 void [applyBeforeApplying](#) ([TridiagonalOperator](#) &) const [virtual]

This method modifies an operator  $L$  before it is applied to an array  $u$  so that  $v = Lu$  will satisfy the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

#### 7.162.2.2 void [applyAfterApplying](#) ([Array](#) &) const [virtual]

This method modifies an array  $u$  so that it satisfies the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

#### 7.162.2.3 void [applyBeforeSolving](#) ([TridiagonalOperator](#) &, [Array](#) & rhs) const [virtual]

This method modifies an operator  $L$  before the linear system  $Lu' = u$  is solved so that  $u'$  will satisfy the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).



**7.162.2.4 void applyAfterSolving ([Array](#) &) const** [virtual]

This method modifies an array  $u$  so that it satisfies the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

**7.162.2.5 void setTime ([Time](#))** [virtual]

This method sets the current time for time-dependent boundary conditions.

Implements [BoundaryCondition< TridiagonalOperator >](#).

## 7.163 Discount Struct Reference

```
#include <ql/TermStructures/bootstraptraits.hpp>
```

### 7.163.1 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)

## 7.164 DiscrepancyStatistics Class Reference

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

Inheritance diagram for DiscrepancyStatistics:



### 7.164.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

It inherit from [SequenceStatistics<Statistics>](#) and adds  $L^2$  discrepancy calculation

#### 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)`

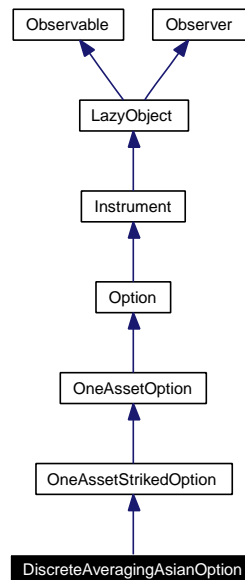
#### 1-dimensional inspectors

- `Real discrepancy () const`

## 7.165 DiscreteAveragingAsianOption Class Reference

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

Inheritance diagram for DiscreteAveragingAsianOption:



### 7.165.1 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.*

## 7.165.2 Member Function Documentation

### 7.165.2.1 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](#).

## 7.166 DiscreteAveragingAsianOption::arguments Class Reference

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

### 7.166.1 Detailed Description

Extra arguments for single-asset discrete-average Asian option.

#### Public Member Functions

- void **validate** () const

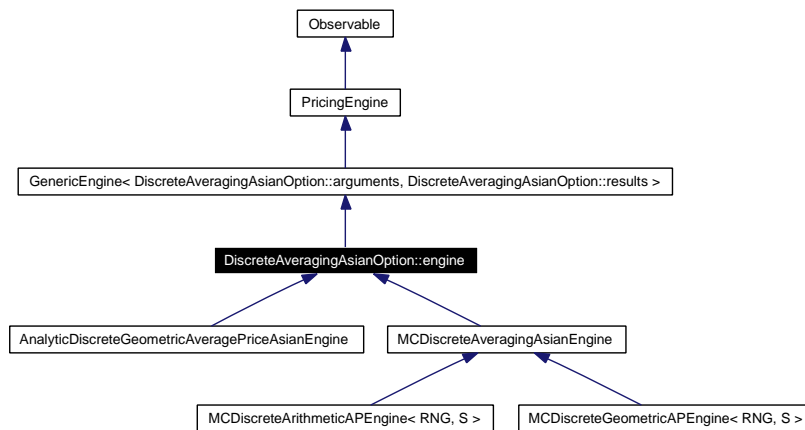
#### Public Attributes

- Average::Type **averageType**
- [Real](#) **runningAccumulator**
- [Size](#) **pastFixings**
- std::vector< [Date](#) > **fixingDates**

## 7.167 DiscreteAveragingAsianOption::engine Class Reference

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

Inheritance diagram for DiscreteAveragingAsianOption::engine:



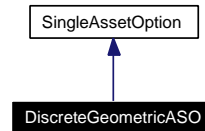
### 7.167.1 Detailed Description

Discrete-averaging Asian engine base class.

## 7.168 DiscreteGeometricASO Class Reference

```
#include <ql/Pricers/discretegeometricaso.hpp>
```

Inheritance diagram for DiscreteGeometricASO:



### 7.168.1 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

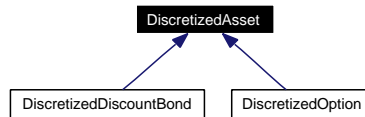
- **DiscreteGeometricASO** (Option::Type type, [Real](#) underlying, [Spread](#) dividendYield, [Rate](#) riskFreeRate, const std::vector< [Time](#) > &times, [Volatility](#) volatility)
- [Real](#) **value** () const
- [Real](#) **delta** () const
- [Real](#) **gamma** () const
- [Real](#) **theta** () const
- boost::shared\_ptr< [SingleAssetOption](#) > **clone** () const



## 7.169 DiscretizedAsset Class Reference

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

Inheritance diagram for DiscretizedAsset:



### 7.169.1 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_`

## 7.169.2 Member Function Documentation

### 7.169.2.1 `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](#).

### 7.169.2.2 `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.

### 7.169.2.3 `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.

### 7.169.2.4 `void adjustValues ()`

This method performs both pre- and post-adjustment

### 7.169.2.5 `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](#).

### 7.169.2.6 `bool isOnTime (Time t) const` [protected]

This method checks whether the asset was rolled at the given time.

**7.169.2.7 virtual void preAdjustValuesImpl ()** [protected, virtual]

This method performs the actual pre-adjustment

**7.169.2.8 virtual void postAdjustValuesImpl ()** [protected, virtual]

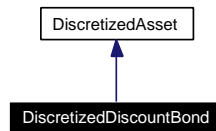
This method performs the actual post-adjustment

Reimplemented in [DiscretizedOption](#).

## 7.170 DiscretizedDiscountBond Class Reference

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

Inheritance diagram for DiscretizedDiscountBond:



### 7.170.1 Detailed Description

Useful discretized discount bond asset.

#### Public Member Functions

- void [reset](#) ([Size](#) size)
- [std::vector](#)< [Time](#) > [mandatoryTimes](#) () const

### 7.170.2 Member Function Documentation

#### 7.170.2.1 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](#).

#### 7.170.2.2 [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:

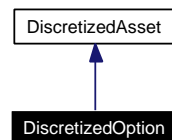
The returned values are not guaranteed to be sorted.

Implements [DiscretizedAsset](#).

## 7.171 DiscretizedOption Class Reference

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

Inheritance diagram for DiscretizedOption:



### 7.171.1 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\\_](#)

### 7.171.2 Member Function Documentation

#### 7.171.2.1 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](#).

**7.171.2.2** `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](#).

**7.171.2.3** `void postAdjustValuesImpl ()` [protected, virtual]

This method performs the actual post-adjustment

Reimplemented from [DiscretizedAsset](#).

## 7.172 Disposable Class Template Reference

```
#include <ql/Utilities/disposable.hpp>
```

### 7.172.1 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;  
}
```

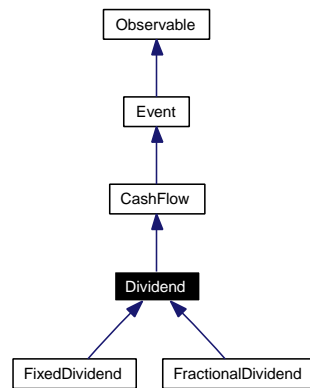
### Public Member Functions

- **Disposable** (`T &t`)
- **Disposable** (`const Disposable< T > &t`)
- **Disposable< T > &operator=** (`const Disposable< T > &t`)

## 7.173 Dividend Class Reference

```
#include <ql/CashFlows/dividend.hpp>
```

Inheritance diagram for Dividend:



### 7.173.1 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
- virtual [Real](#) **amount** () const =0  
*returns the amount of the cash flow*

#### Visitability

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

### Protected Attributes

- [Date](#) date\_

### 7.173.2 Member Function Documentation

#### 7.173.2.1 virtual [Date](#) date () const [virtual]

#### Note:

This is inherited from the event class



Implements [CashFlow](#).

7.173.2.2 **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.

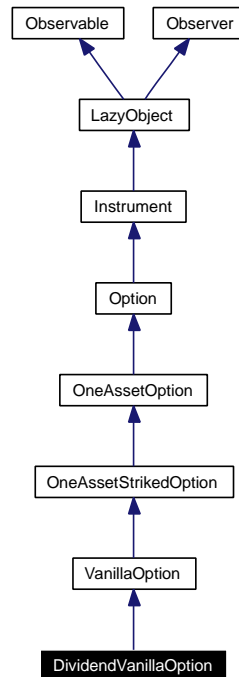
Implements [CashFlow](#).

Implemented in [FixedDividend](#), and [FractionalDividend](#).

## 7.174 DividendVanillaOption Class Reference

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

Inheritance diagram for DividendVanillaOption:



### 7.174.1 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.

## 7.174.2 Member Function Documentation

### 7.174.2.1 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](#).

## 7.175 DividendVanillaOption::arguments Class Reference

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

### 7.175.1 Detailed Description

Arguments for dividend vanilla option calculation

#### Public Member Functions

- void **validate** () const

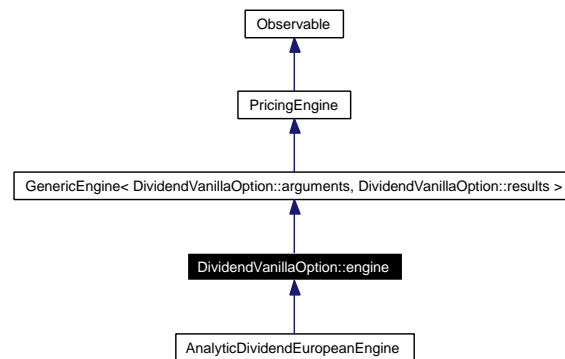
#### Public Attributes

- DividendSchedule **cashFlow**

## 7.176 DividendVanillaOption::engine Class Reference

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

Inheritance diagram for DividendVanillaOption::engine:



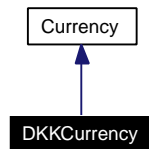
### 7.176.1 Detailed Description

[Dividend](#) vanilla option engine base class.

## 7.177 DKKCurrency Class Reference

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

Inheritance diagram for DKKCurrency:



### 7.177.1 Detailed Description

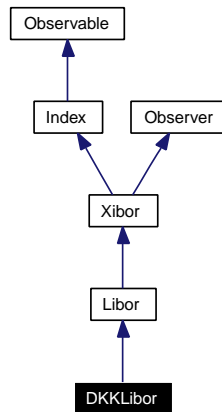
Danish krone.

The ISO three-letter code is DKK; the numeric code is 208. It is divided in 100 øre.

## 7.178 DKKLabor Class Reference

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

Inheritance diagram for DKKLabor:



### 7.178.1 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>>.

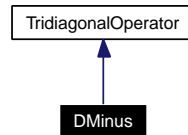
### Public Member Functions

- **DKKLabor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.179 DMinus Class Reference

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

Inheritance diagram for DMinus:



### 7.179.1 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

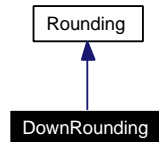
- **DMinus** ([Size](#) gridPoints, [Real](#) h)



## 7.180 DownRounding Class Reference

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

Inheritance diagram for DownRounding:



### 7.180.1 Detailed Description

Down-rounding.

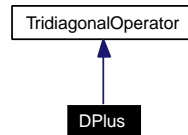
#### Public Member Functions

- **DownRounding** ([Integer](#) precision, [Integer](#) digit=5)

## 7.181 DPlus Class Reference

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

Inheritance diagram for DPlus:



### 7.181.1 Detailed Description

$D_+$  matricial representation

The differential operator  $D_+$  discretizes the first derivative with the first-order formula

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

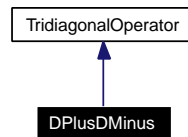
### Public Member Functions

- **DPlus** ([Size](#) gridPoints, [Real](#) h)

## 7.182 DPlusDMinus Class Reference

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

Inheritance diagram for DPlusDMinus:



### 7.182.1 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.

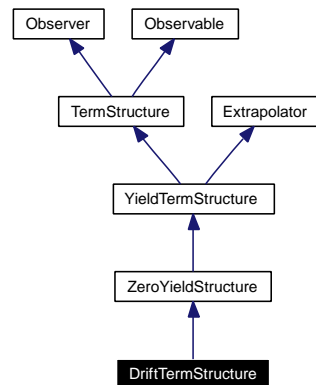
### Public Member Functions

- **DPlusDMinus** ([Size](#) gridPoints, [Real](#) h)

## 7.183 DriftTermStructure Class Reference

```
#include <ql/TermStructures/drifttermstructure.hpp>
```

Inheritance diagram for DriftTermStructure:



### 7.183.1 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 reference date, i.e., the date at which discount = 1*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return rates*

## Protected Member Functions

- [Rate zeroYieldImpl](#) ([Time](#)) const  
*returns the discount factor as seen from the evaluation date*

## 7.184 Duration Struct Reference

```
#include <ql/CashFlows/analysis.hpp>
```

### 7.184.1 Detailed Description

duration type

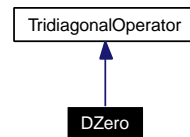
#### Public Types

- enum Type { Simple, Macaulay, Modified }

## 7.185 DZero Class Reference

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

Inheritance diagram for DZero:



### 7.185.1 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.

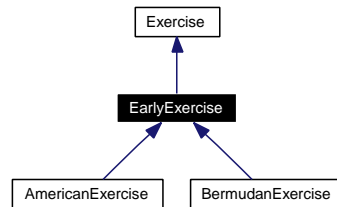
### Public Member Functions

- **DZero** ([Size](#) gridPoints, [Real](#) h)

## 7.186 EarlyExercise Class Reference

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

Inheritance diagram for EarlyExercise:



### 7.186.1 Detailed Description

Early-exercise base class.

The payoff can be at exercise (default case) or at expiry

#### Todo

derive a plain American [Exercise](#) class (no `earliestDate`, no `payoffAtExpiry`)

### Public Member Functions

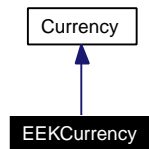
- **EarlyExercise** (Type type=Undefined, bool payoffAtExpiry=false)
- bool **payoffAtExpiry** () const



## 7.187 EEKCurrency Class Reference

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

Inheritance diagram for EEKCurrency:



### 7.187.1 Detailed Description

Estonian kroon.

The ISO three-letter code is EEK; the numeric code is 233. It is divided in 100 senti.

## 7.188 EndCriteria Class Reference

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

### 7.188.1 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 `checkIterationNumber` ([Size](#) iteration)
- bool `checkStationaryValue` ([Real](#) fold, [Real](#) fnew)
- bool `checkAccuracyValue` ([Real](#) f)
- bool `checkStationaryGradientNorm` ([Real](#) normDiff)
- bool `checkAccuracyGradientNorm` ([Real](#) norm)
- 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*
- Type `criteria` () const  
*return the end criteria type*

### Protected Attributes

- [Size](#) `maxIteration_`  
*Maximum number of iterations.*
- [Real](#) `functionEpsilon_`  
*function and gradient epsilons*
- [Real](#) `gradientEpsilon_`
- [Size](#) `maxIterStatPt_`

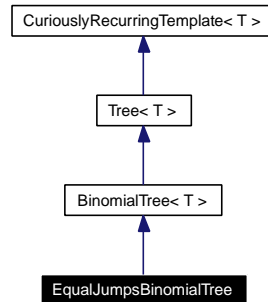
*Maximum number of iterations in stationary state.*

- [Size](#) `statState_`
- Type `endCriteria_`
- bool `positiveOptimization_`

## 7.189 EqualJumpsBinomialTree Class Template Reference

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

Inheritance diagram for EqualJumpsBinomialTree:



### 7.189.1 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

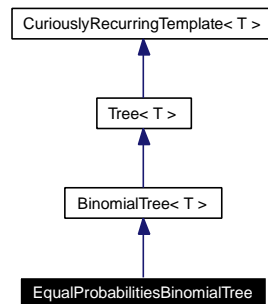
#### Protected Attributes

- **Real dx\_**
- **Real pu\_**
- **Real pd\_**

## 7.190 EqualProbabilitiesBinomialTree Class Template Reference

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

Inheritance diagram for EqualProbabilitiesBinomialTree:



### 7.190.1 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_`

## 7.191 Error Class Reference

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

### 7.191.1 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.*

### 7.191.2 Constructor & Destructor Documentation

**7.191.2.1 [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.

**7.191.2.2 [~Error](#) () throw ()**

the automatically generated destructor would not have the throw specifier.

## 7.192 ErrorFunction Class Reference

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

### 7.192.1 Detailed Description

Error function

formula here ... Used to calculate the cumulative normal distribution function

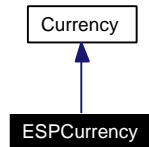
### Public Member Functions

- [Real](#) operator() ([Real](#) x) const

## 7.193 ESPCurrency Class Reference

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

Inheritance diagram for ESPCurrency:



### 7.193.1 Detailed Description

Spanish peseta.

The ISO three-letter code was ESP; the numeric code was 724. It was divided in 100 centimos.

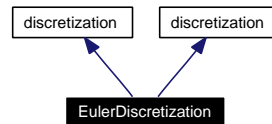
Obsoleted by the Euro since 1999.



## 7.194 EulerDiscretization Class Reference

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

Inheritance diagram for EulerDiscretization:



### 7.194.1 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

### 7.194.2 Member Function Documentation

**7.194.2.1** [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](#).

**7.194.2.2** [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](#).

**7.194.2.3** [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](#).

**7.194.2.4** [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](#).

**7.194.2.5** [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](#).

**7.194.2.6** [Real](#) **variance** (const [StochasticProcess1D](#) &, [Time](#) *t0*, [Real](#) *x0*, [Time](#) *dt*) const  
[virtual]

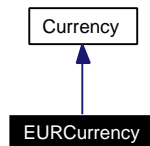
Returns an approximation of the variance defined as  $\sigma(t_0, x_0)^2 \Delta t$ .

Implements [StochasticProcess1D::discretization](#).

## 7.195 EURCurrency Class Reference

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

Inheritance diagram for EURCurrency:



### 7.195.1 Detailed Description

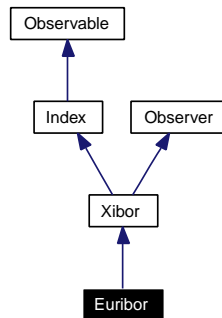
European Euro.

The ISO three-letter code is EUR; the numeric code is 978. It is divided into 100 cents.

## 7.196 Euribor Class Reference

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

Inheritance diagram for Euribor:



### 7.196.1 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.

#### Examples:

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

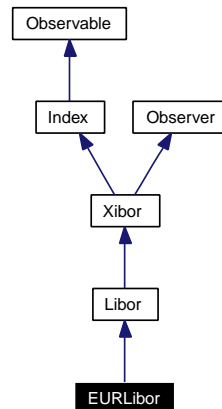
### Public Member Functions

- **Euribor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.197 EURLibor Class Reference

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

Inheritance diagram for EURLibor:



### 7.197.1 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** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.198 EuropeanExercise Class Reference

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

Inheritance diagram for EuropeanExercise:



### 7.198.1 Detailed Description

European exercise.

A European option can only be exercised at one (expiry) date.

**Examples:**

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

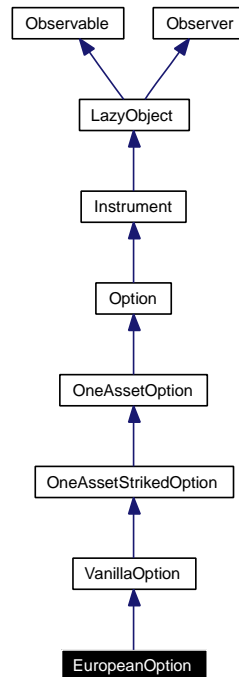
### Public Member Functions

- `EuropeanExercise` (const [Date](#) &date)

## 7.199 EuropeanOption Class Reference

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

Inheritance diagram for EuropeanOption:



### 7.199.1 Detailed Description

European option on a single asset.

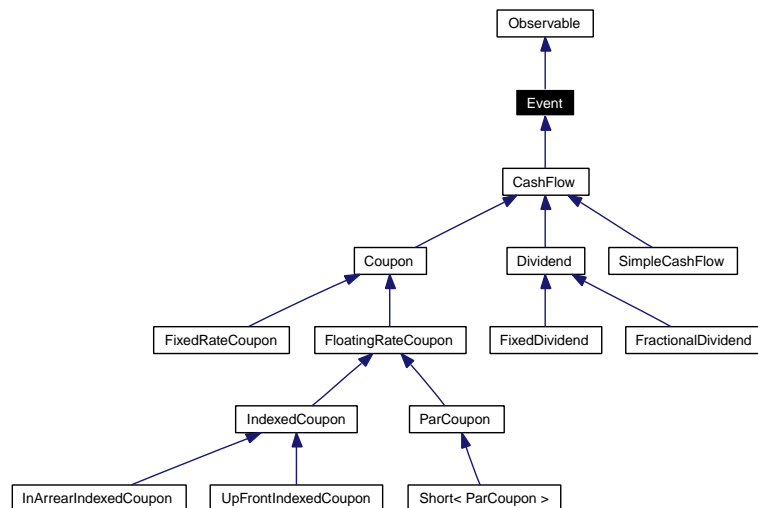
#### Public Member Functions

- **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](#) >())

## 7.200 Event Class Reference

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

Inheritance diagram for Event:



### 7.200.1 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](#) &)

### 7.200.2 Member Function Documentation

#### 7.200.2.1 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?

## 7.201 ExchangeRate Class Reference

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

### 7.201.1 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

#### Constructors

- [ExchangeRate](#) (const [Currency](#) &source, const [Currency](#) &target, [Decimal](#) rate)

#### 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)*

## 7.201.2 Member Enumeration Documentation

### 7.201.2.1 enum [Type](#)

#### Enumerator:

*Direct* given directly by the user

*Derived* derived from exchange rates between other currencies

## 7.201.3 Constructor & Destructor Documentation

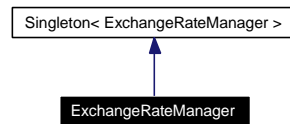
### 7.201.3.1 [ExchangeRate](#) (const [Currency](#) & *source*, const [Currency](#) & *target*, [Decimal](#) *rate*)

the rate  $r$  is given with the convention that a unit of the source is worth  $r$  units of the target.

## 7.202 ExchangeRateManager Class Reference

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

Inheritance diagram for ExchangeRateManager:



### 7.202.1 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](#) >

### 7.202.2 Member Function Documentation

**7.202.2.1** 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.

7.202.2.2 **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.

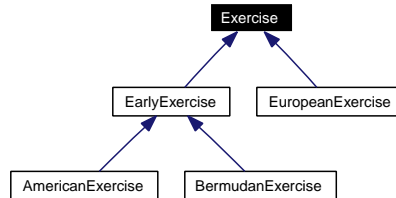
**Warning**

if two or more exchange-rate chains are possible which allow to specify a requested rate, it is unspecified which one is returned.

## 7.203 Exercise Class Reference

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

Inheritance diagram for Exercise:



### 7.203.1 Detailed Description

Base exercise class.

#### Public Types

- enum Type { Undefined = -1, American, Bermudan, European }

#### Public Member Functions

- Exercise (Type type=Undefined)
- bool isNull () const
- Type type () const
- Date date (Size index) const
- const std::vector< Date > & dates () const
- Date lastDate () const

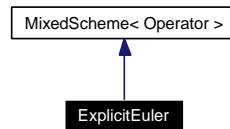
#### Protected Attributes

- std::vector< Date > dates\_
- Type type\_

## 7.204 ExplicitEuler Class Template Reference

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

Inheritance diagram for ExplicitEuler:



### 7.204.1 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

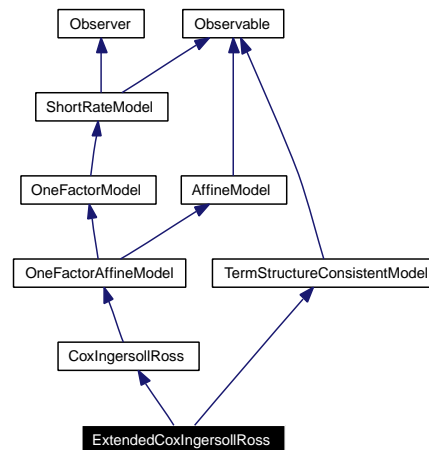
- **ExplicitEuler** (const operator\_type &L, const std::vector< boost::shared\_ptr< bc\_type > > &bcs)



## 7.205 ExtendedCoxIngersollRoss Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss:



### 7.205.1 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

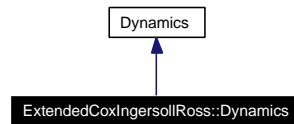
## Classes

- class [Dynamics](#)  
*Short-rate dynamics in the extended Cox-Ingersoll-Ross model.*
- class [FittingParameter](#)  
*Analytical term-structure fitting parameter  $\varphi(t)$ .*

## 7.206 ExtendedCoxIngersollRoss::Dynamics Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss::Dynamics:



### 7.206.1 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.

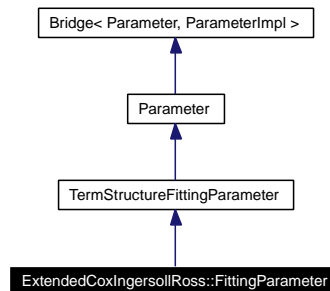
### Public Member Functions

- **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

## 7.207 ExtendedCoxIngersollRoss::FittingParameter Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss::FittingParameter:



### 7.207.1 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  $h = \sqrt{k^2 + 2\sigma^2}$ .

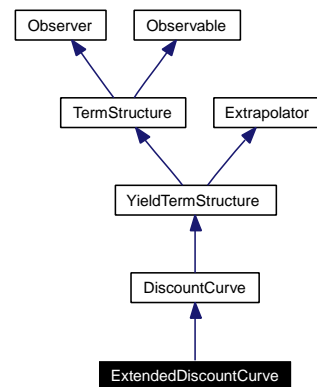
### Public Member Functions

- **FittingParameter** (const [Handle< YieldTermStructure >](#) &termStructure, [Real](#) theta, [Real](#) k, [Real](#) sigma, [Real](#) x0)

## 7.208 ExtendedDiscountCurve Class Reference

```
#include <ql/TermStructures/extendeddiscountcurve.hpp>
```

Inheritance diagram for ExtendedDiscountCurve:



### 7.208.1 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

## 7.208.2 Member Function Documentation

### 7.208.2.1 `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](#).

### 7.208.2.2 `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.

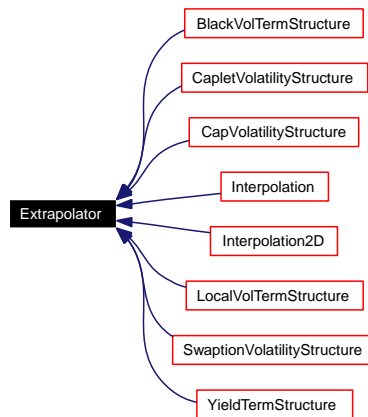
### 7.208.2.3 `Rate zeroYieldImpl (Time) const` [protected]

Returns the zero yield rate for the given date calculating it from the discount.

## 7.209 Extrapolator Class Reference

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

Inheritance diagram for Extrapolator:



### 7.209.1 Detailed Description

base class for classes possibly allowing extrapolation

#### Public Member Functions

##### modifiers

- void [enableExtrapolation](#) ()  
*enable extrapolation in subsequent calls*
- void [disableExtrapolation](#) ()  
*disable extrapolation in subsequent calls*

##### inspectors

- bool [allowsExtrapolation](#) () const  
*tells whether extrapolation is enabled*

## 7.210 Factorial Class Reference

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

### 7.210.1 Detailed Description

Factorial numbers calculator

#### Tests

the correctness of the returned value is tested by checking it against numerical calculations.

### Static Public Member Functions

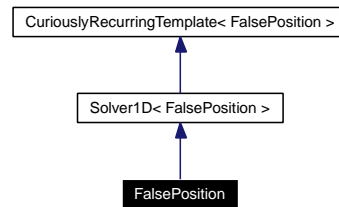
- static [Real](#) [get](#) ([Natural](#) n)
- static [Real](#) [ln](#) ([Natural](#) n)



## 7.211 FalsePosition Class Reference

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

Inheritance diagram for FalsePosition:



### 7.211.1 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`

## 7.212 FaureRsg Class Reference

```
#include <ql/RandomNumbers/faurersg.hpp>
```

### 7.212.1 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

## 7.213 FDAmericanCondition Class Template Reference

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

### 7.213.1 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)

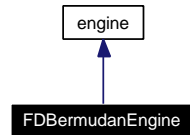
### Protected Member Functions

- void **initializeStepCondition** () const

## 7.214 FDBermudanEngine Class Reference

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

Inheritance diagram for FDBermudanEngine:



### 7.214.1 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

## 7.215 FDDividendEngineMerton73 Class Reference

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

### 7.215.1 Detailed Description

Finite-differences pricing engine for dividend options using.

### Public Member Functions

- **FDDividendEngineMerton73** ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool timeDependent=false)

## 7.216 FDDividendEngineShiftScale Class Reference

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

### 7.216.1 Detailed Description

Finite-differences pricing engine for dividend options using.

#### Public Member Functions

- **FDDividendEngineShiftScale** ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool time-Dependent=false)

## 7.217 FDEuropeanEngine Class Reference

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

### 7.217.1 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)

## 7.218 FDStepConditionEngine Class Reference

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

### 7.218.1 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

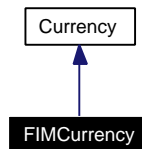
- boost::shared\_ptr< [StandardStepCondition](#) > **stepCondition\_**
- [SampledCurve](#) **prices\_**
- [TridiagonalOperator](#) **controlOperator\_**
- std::vector< boost::shared\_ptr< bc\_type > > **controlBCs\_**
- [SampledCurve](#) **controlPrices\_**



## 7.219 FIMCurrency Class Reference

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

Inheritance diagram for FIMCurrency:



### 7.219.1 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.

## 7.220 FiniteDifferenceModel Class Template Reference

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

### 7.220.1 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)

### 7.220.2 Member Function Documentation

#### 7.220.2.1 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.

#### 7.220.2.2 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.

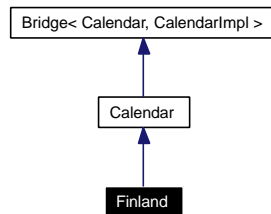
#### [Warning](#)

being this a rollback, from must be a later time than to.

## 7.221 Finland Class Reference

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

Inheritance diagram for Finland:



### 7.221.1 Detailed Description

Finnish calendar.

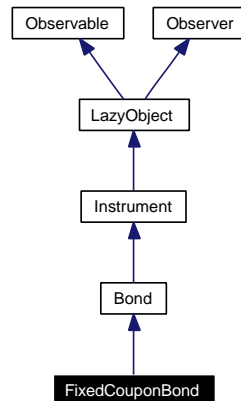
Holidays:

- 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

## 7.222 FixedCouponBond Class Reference

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

Inheritance diagram for FixedCouponBond:



### 7.222.1 Detailed Description

fixed-coupon bond

#### Tests

calculations are tested by checking results against cached values.

### Public Member Functions

- **FixedCouponBond** (const [Date](#) &issueDate, const [Date](#) &datedDate, const [Date](#) &maturityDate, [Integer](#) settlementDays, const std::vector< [Rate](#) > &coupons, [Frequency](#) couponFrequency, const [DayCounter](#) &dayCounter, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention=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)

## 7.222.2 Constructor & Destructor Documentation

7.222.2.1 **FixedCouponBond** (const **Date** & *issueDate*, const **Date** & *datedDate*, const **Date** & *maturityDate*, **Integer** *settlementDays*, const std::vector< **Rate** > & *coupons*, **Frequency** *couponFrequency*, const **DayCounter** & *dayCounter*, const **Calendar** & *calendar*, **BusinessDayConvention** *convention* = Following, **Real** *redemption* = 100.0, const **Handle**< **YieldTermStructure** > & *discountCurve* = **Handle**< **YieldTermStructure** >(), const **Date** & *stub* = **Date**(), bool *fromEnd* = true, bool *longFinal* = false)

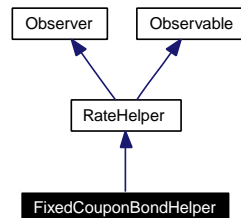
### Deprecated

use the other constructor

## 7.223 FixedCouponBondHelper Class Reference

```
#include <ql/TermStructures/bondhelpers.hpp>
```

Inheritance diagram for FixedCouponBondHelper:



### 7.223.1 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 [DayCounter](#) &dayCounter, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention=Following, [Real](#) redemption=100.0, const [Date](#) &stub=[Date](#)(), bool fromEnd=true)
- **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_`

## 7.223.2 Constructor & Destructor Documentation

- 7.223.2.1 **[FixedCouponBondHelper](#)** (`const Handle< Quote > & cleanPrice`, `const Date & issueDate`, `const Date & datedDate`, `const Date & maturityDate`, `Integer settlementDays`, `const std::vector< Rate > & coupons`, `Frequency frequency`, `const DayCounter & dayCounter`, `const Calendar & calendar`, `BusinessDayConvention convention = Following`, `Real redemption = 100.0`, `const Date & stub = Date()`, `bool fromEnd = true`)

### Deprecated

use the other constructor

## 7.223.3 Member Function Documentation

- 7.223.3.1 **[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.

Implements [RateHelper](#).

- 7.223.3.2 **`void setTermStructure (YieldTermStructure *)`** [virtual]

sets the term structure to be used for pricing

### Warning

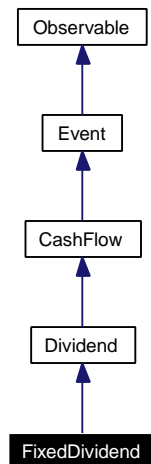
Being a pointer and not a `shared_ptr`, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

## 7.224 FixedDividend Class Reference

```
#include <ql/CashFlows/dividend.hpp>
```

Inheritance diagram for FixedDividend:



### 7.224.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

#### Public Member Functions

- **FixedDividend** ([Real](#) amount, const [Date](#) &date)

##### CashFlow interface

- virtual [Real](#) **amount** () const  
*returns the amount of the cash flow*
- virtual [Real](#) **amount** ([Real](#) underlying) const

#### Protected Attributes

- [Real](#) **amount\_**

### 7.224.2 Member Function Documentation

#### 7.224.2.1 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.

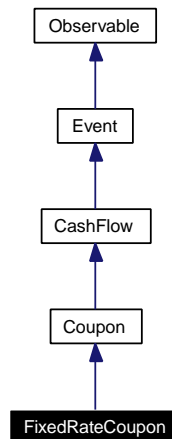


Implements [Dividend](#).

## 7.225 FixedRateCoupon Class Reference

```
#include <ql/CashFlows/fixedratecoupon.hpp>
```

Inheritance diagram for FixedRateCoupon:



### 7.225.1 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](#) &)

## 7.225.2 Member Function Documentation

### 7.225.2.1 [Real](#) amount () const [virtual]

returns the amount of the cash flow

**Note:**

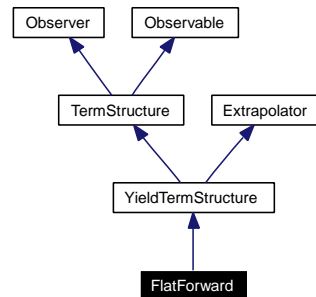
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

## 7.226 FlatForward Class Reference

```
#include <ql/TermStructures/flatforward.hpp>
```

Inheritance diagram for FlatForward:



### 7.226.1 Detailed Description

Flat interest-rate curve.

Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), [DiscreteHedging.cpp](#), and [Equity-Option.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 rates*
- void **update** ()

## 7.226.2 Member Function Documentation

### 7.226.2.1 void update () [virtual]

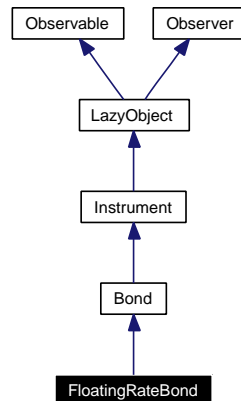
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

## 7.227 FloatingRateBond Class Reference

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

Inheritance diagram for FloatingRateBond:



### 7.227.1 Detailed Description

floating-rate bond

#### Tests

calculations are tested by checking results against cached values.

### Public Member Functions

- **FloatingRateBond** (const [Date](#) &issueDate, const [Date](#) &datedDate, const [Date](#) &maturityDate, [Integer](#) settlementDays, const boost::shared\_ptr< [Xibor](#) > &index, [Integer](#) fixingDays, const std::vector< [Spread](#) > &spreads, [Frequency](#) couponFrequency, const [DayCounter](#) &dayCounter, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention=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< [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)

## 7.227.2 Constructor & Destructor Documentation

7.227.2.1 **FloatingRateBond** (const **Date** & *issueDate*, const **Date** & *datedDate*, const **Date** & *maturityDate*, **Integer** *settlementDays*, const boost::shared\_ptr< **Xibor** > & *index*, **Integer** *fixingDays*, const std::vector< **Spread** > & *spreads*, **Frequency** *couponFrequency*, const **DayCounter** & *dayCounter*, const **Calendar** & *calendar*, **BusinessDayConvention** *convention* = Following, **Real** *redemption* = 100.0, const **Handle**< **YieldTermStructure** > & *discountCurve* = **Handle**< **YieldTermStructure** >(), const **Date** & *stub* = **Date**(), bool *fromEnd* = true)

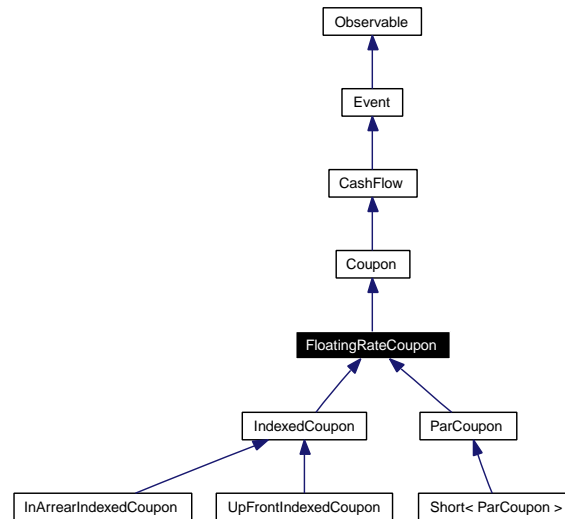
### Deprecated

use the other constructor

## 7.228 FloatingRateCoupon Class Reference

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

Inheritance diagram for FloatingRateCoupon:



### 7.228.1 Detailed Description

Coupon paying a variable 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.

### Public Member Functions

- **FloatingRateCoupon** ([Real](#) nominal, const [Date](#) &paymentDate, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)())

#### Coupon interface

- [Rate](#) **rate** () const  
*accrued rate*
- [Real](#) **accruedAmount** (const [Date](#) &) const  
*accrued amount at the given date*

#### Inspectors

- [Integer](#) **fixingDays** () const  
*fixing days*



- virtual [Spread](#) `spread` () const  
*spread paid over the fixing of the underlying index*
- virtual [Rate indexFixing](#) () const =0  
*fixing of the underlying index*
- virtual [Date fixingDate](#) () const =0  
*fixing date*

### Visitability

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

### Protected Member Functions

- virtual [Rate convexityAdjustment](#) ([Rate](#) fixing) const  
*convexity adjustment for the given index fixing*

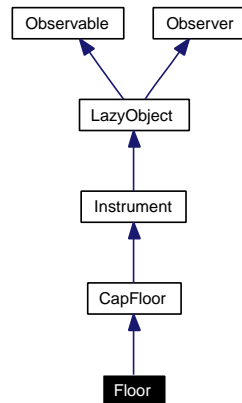
### Protected Attributes

- [Integer](#) `fixingDays_`
- [Spread](#) `spread_`

## 7.229 Floor Class Reference

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

Inheritance diagram for Floor:



### 7.229.1 Detailed Description

Concrete floor class.

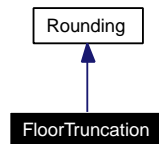
#### Public Member Functions

- **Floor** (const std::vector< boost::shared\_ptr< [CashFlow](#) > > &floatingLeg, const std::vector< [Rate](#) > &exerciseRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared\_ptr< [PricingEngine](#) > &engine)

## 7.230 FloorTruncation Class Reference

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

Inheritance diagram for FloorTruncation:



### 7.230.1 Detailed Description

[Floor](#) truncation.

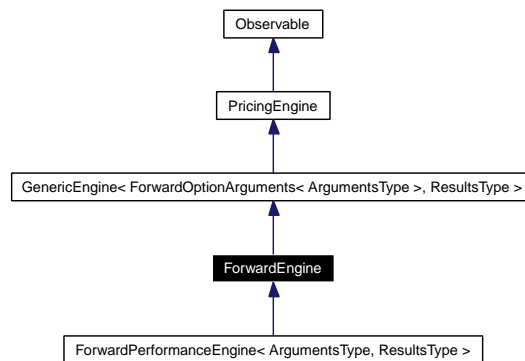
#### Public Member Functions

- **FloorTruncation** ([Integer](#) precision, [Integer](#) digit=5)

## 7.231 ForwardEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

Inheritance diagram for ForwardEngine:



### 7.231.1 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\_**

## 7.232 ForwardFlat Class Reference

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

### 7.232.1 Detailed Description

Forward-flat interpolation factory and traits.

#### Public Types

- enum { **global** = 0 }

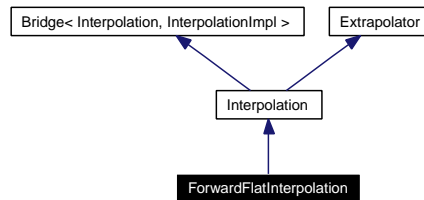
#### Public Member Functions

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

## 7.233 ForwardFlatInterpolation Class Reference

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

Inheritance diagram for ForwardFlatInterpolation:



### 7.233.1 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)`

### 7.233.2 Constructor & Destructor Documentation

#### 7.233.2.1 [ForwardFlatInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

**Precondition:**

the *x* values must be sorted.

## 7.234 ForwardOptionArguments Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

### 7.234.1 Detailed Description

```
template<class ArgumentsType> class QuantLib::ForwardOptionArguments< Arguments-  
Type >
```

Arguments for forward (strike-resetting) option calculation

### Public Member Functions

- `void validate () const`

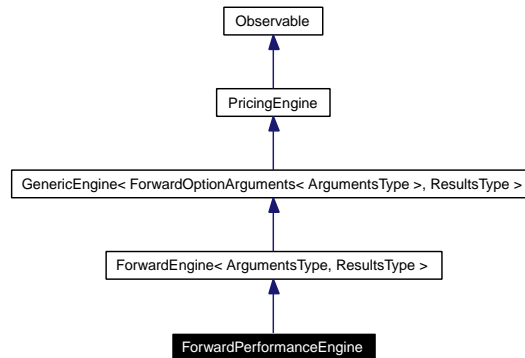
### Public Attributes

- [Real](#) `moneyness`
- [Date](#) `resetDate`

## 7.235 ForwardPerformanceEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardperformanceengine.hpp>
```

Inheritance diagram for ForwardPerformanceEngine:



### 7.235.1 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



## 7.236 ForwardRate Struct Reference

```
#include <ql/TermStructures/bootstraptraits.hpp>
```

### 7.236.1 Detailed Description

Forward-curve traits.

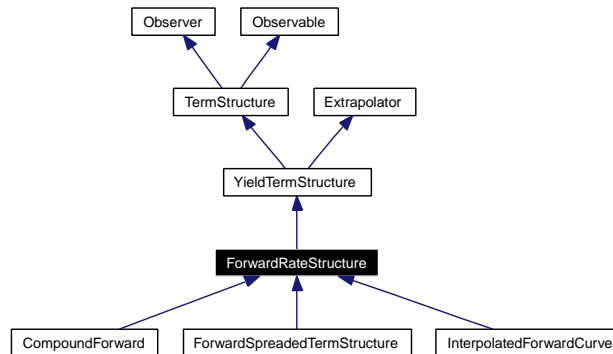
#### Static Public Member Functions

- 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)

## 7.237 ForwardRateStructure Class Reference

```
#include <ql/TermStructures/forwardstructure.hpp>
```

Inheritance diagram for ForwardRateStructure:



### 7.237.1 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.

### Public Member Functions

#### Constructors

See the [TermStructure](#) documentation for issues regarding constructors.

- [ForwardRateStructure](#) ()
- [ForwardRateStructure](#) (const [Date](#) &referenceDate)
- [ForwardRateStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)

### Protected Member Functions

#### YieldTermStructure implementation

- [DiscountFactor](#) `discountImpl` ([Time](#)) const
- virtual [Rate](#) `forwardImpl` ([Time](#)) const =0  
*instantaneous forward-rate calculation*
- virtual [Rate](#) `zeroYieldImpl` ([Time](#)) const

### 7.237.2 Member Function Documentation

#### 7.237.2.1 [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](#).

#### 7.237.2.2 [Rate](#) zeroYieldImpl ([Time](#)) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

##### **Warning**

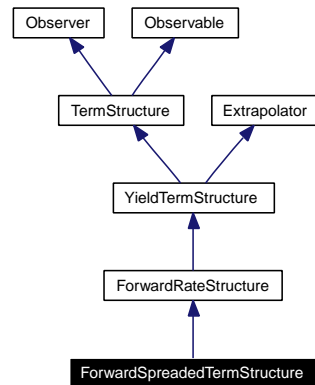
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Reimplemented in [CompoundForward](#), [InterpolatedForwardCurve](#), and [ForwardSpreadedTermStructure](#).

## 7.238 ForwardSpreadedTermStructure Class Reference

```
#include <ql/TermStructures/forwardspreadedtermstructure.hpp>
```

Inheritance diagram for ForwardSpreadedTermStructure:



### 7.238.1 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 reference date, i.e., the date at which discount = 1*
- [Date](#) [maxDate](#) () const

*the latest date for which the curve can return rates*

- [Time maxTime](#) () const  
*the latest time for which the curve can return rates*

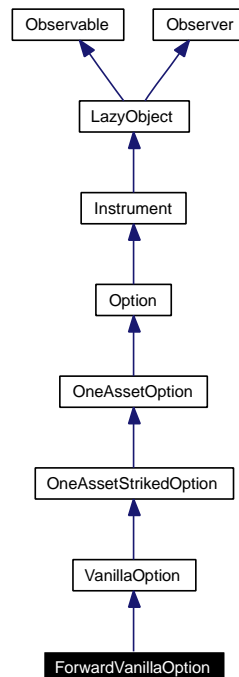
### Protected Member Functions

- [Rate forwardImpl](#) (Time) const  
*returns the spreaded forward rate*
- [Rate zeroYieldImpl](#) (Time) const  
*returns the spreaded zero yield rate*

## 7.239 ForwardVanillaOption Class Reference

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

Inheritance diagram for ForwardVanillaOption:



### 7.239.1 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

## 7.239.2 Member Function Documentation

### 7.239.2.1 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](#).

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

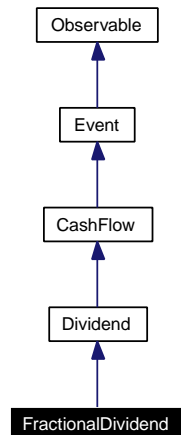
When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetStrikedOption](#).

## 7.240 FractionalDividend Class Reference

```
#include <ql/CashFlows/dividend.hpp>
```

Inheritance diagram for FractionalDividend:



### 7.240.1 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)

#### CashFlow interface

- virtual [Real](#) **amount** () const  
*returns the amount of the cash flow*
- virtual [Real](#) **amount** ([Real](#) underlying) const
- virtual [Real](#) **rate** () const
- virtual [Real](#) **nominal** () const

### Protected Attributes

- [Real](#) rate\_
- [Real](#) nominal\_

### 7.240.2 Member Function Documentation

#### 7.240.2.1 virtual [Real](#) amount () const [virtual]

returns the amount of the cash flow



**Note:**

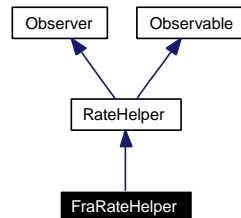
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [Dividend](#).

## 7.241 FraRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for FraRateHelper:



### 7.241.1 Detailed Description

Forward rate agreement helper.

#### Warning

This class assumes that the reference date does not change between calls of `setTermStructure()`.

#### Todo

convexity adjustment should be implemented.

#### Examples:

[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
- void **setTermStructure** ([YieldTermStructure](#) \*)  
*sets the term structure to be used for pricing*
- **Date latestDate** () const  
*latest relevant date*

### 7.241.2 Member Function Documentation

#### 7.241.2.1 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](#).

**7.241.2.2 [Date](#) latestDate () const [virtual]**

latest relevant date

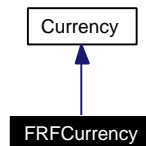
The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

## 7.242 FRFCurrency Class Reference

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

Inheritance diagram for FRFCurrency:



### 7.242.1 Detailed Description

French franc.

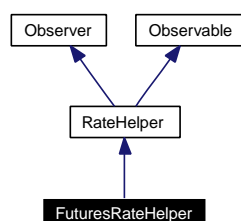
The ISO three-letter code was FRF; the numeric code was 250. It was divided in 100 centimes.

Obsoleted by the Euro since 1999.

## 7.243 FuturesRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for FuturesRateHelper:



### 7.243.1 Detailed Description

Interest-rate futures helper.

#### Warning

This class assumes that the reference date does not change between calls of `setTermStructure()`.

#### Examples:

`swapvaluation.cpp`.

### Public Member Functions

- **FuturesRateHelper** (const [Handle](#)< [Quote](#) > &price, const [Date](#) &immDate, [Integer](#) nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **FuturesRateHelper** (const [Handle](#)< [Quote](#) > &price, const [Date](#) &immDate, const [Date](#) &matDate, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **FuturesRateHelper** ([Real](#) price, const [Date](#) &immDate, [Integer](#) nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **Real impliedQuote** () const
- **DiscountFactor** discountGuess () const
- **Date** latestDate () const  
*latest relevant date*

### 7.243.2 Member Function Documentation

#### 7.243.2.1 [Date](#) latestDate () const [virtual]

latest relevant date

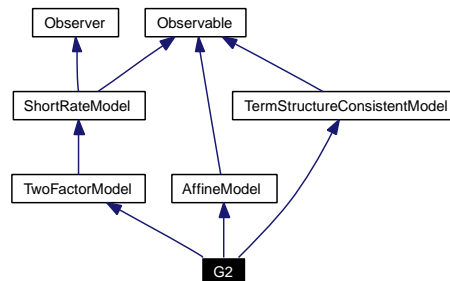
The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

## 7.244 G2 Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Inheritance diagram for G2:



### 7.244.1 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**

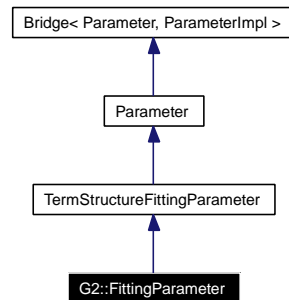
## Classes

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

## 7.245 G2::FittingParameter Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Inheritance diagram for G2::FittingParameter:



### 7.245.1 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

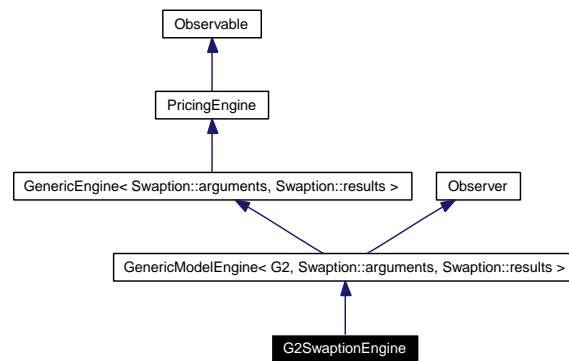
- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a, [Real](#) sigma, [Real](#) b, [Real](#) eta, [Real](#) rho)



## 7.246 G2SwaptionEngine Class Reference

```
#include <ql/PricingEngines/Swaption/g2swaptionengine.hpp>
```

Inheritance diagram for G2SwaptionEngine:



### 7.246.1 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

## 7.247 GammaFunction Class Reference

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

### 7.247.1 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.

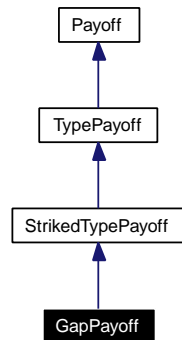
### Public Member Functions

- [Real](#) logValue ([Real](#) x) const

## 7.248 GapPayoff Class Reference

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

Inheritance diagram for GapPayoff:



### 7.248.1 Detailed Description

Binary gap payoff.

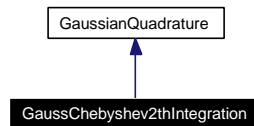
#### Public Member Functions

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

## 7.249 GaussChebyshev2thIntegration Class Reference

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

Inheritance diagram for GaussChebyshev2thIntegration:



### 7.249.1 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

$$w(x) = (1 - x^2)^{1/2}$$

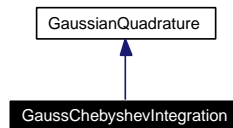
### Public Member Functions

- `GaussChebyshev2thIntegration` ([Size](#) n)

## 7.250 GaussChebyshevIntegration Class Reference

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

Inheritance diagram for GaussChebyshevIntegration:



### 7.250.1 Detailed Description

Gauss-Chebyshev integration.

This class performs a 1-dimensional Gauss-Chebyshev integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{-1/2}$$

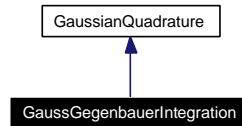
### Public Member Functions

- `GaussChebyshevIntegration` ([Size](#) n)

## 7.251 GaussGegenbauerIntegration Class Reference

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

Inheritance diagram for GaussGegenbauerIntegration:



### 7.251.1 Detailed Description

Gauss-Gegenbauer integration.

This class performs a 1-dimensional Gauss-Gegenbauer integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{\lambda-1/2}$$

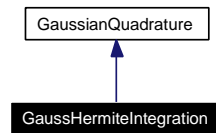
### Public Member Functions

- `GaussGegenbauerIntegration` ([Size](#) n, [Real](#) lambda)

## 7.252 GaussHermiteIntegration Class Reference

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

Inheritance diagram for GaussHermiteIntegration:



### 7.252.1 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

$$\mu > -0.5$$

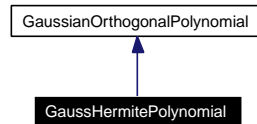
### Public Member Functions

- `GaussHermiteIntegration` ([Size](#) n, [Real](#) mu=0.0)

## 7.253 GaussHermitePolynomial Class Reference

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

Inheritance diagram for GaussHermitePolynomial:



### 7.253.1 Detailed Description

Gauss-Hermite polynomial.

#### Public Member Functions

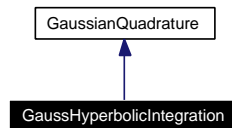
- **GaussHermitePolynomial** ([Real](#) mu=0.0)
- [Real](#) **mu\_0** () const
- [Real](#) **alpha** ([Size](#) i) const
- [Real](#) **beta** ([Size](#) i) const
- [Real](#) **w** ([Real](#) x) const



## 7.254 GaussHyperbolicIntegration Class Reference

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

Inheritance diagram for GaussHyperbolicIntegration:



### 7.254.1 Detailed Description

Gauss-Hyperbolic integration.

This class performs a 1-dimensional Gauss-Hyperbolic integration.

$$\int_{-\infty}^{\infty} f(x) dx$$

The weighting function is

$$w(x) = 1/\cosh(x)$$

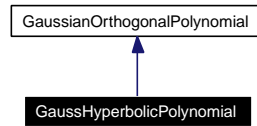
### Public Member Functions

- `GaussHyperbolicIntegration` ([Size n](#))

## 7.255 GaussHyperbolicPolynomial Class Reference

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

Inheritance diagram for GaussHyperbolicPolynomial:



### 7.255.1 Detailed Description

Gauss hyperbolic polynomial.

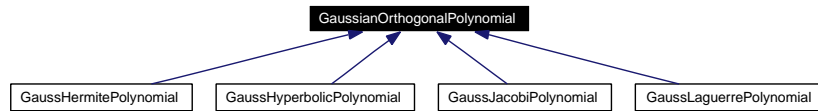
#### Public Member Functions

- [Real](#) **mu\_0** () const
- [Real](#) **alpha** ([Size](#) i) const
- [Real](#) **beta** ([Size](#) i) const
- [Real](#) **w** ([Real](#) x) const

## 7.256 GaussianOrthogonalPolynomial Class Reference

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

Inheritance diagram for GaussianOrthogonalPolynomial:



### 7.256.1 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

$$\mu_0 = \int w(x)dx$$

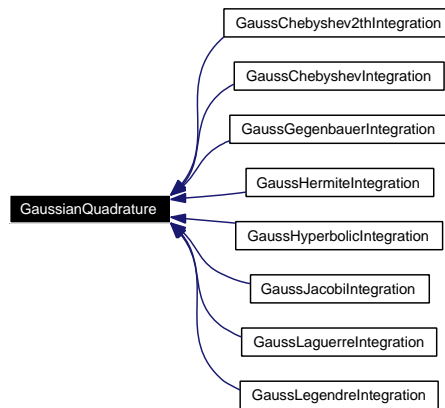
### Public Member Functions

- 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

## 7.257 GaussianQuadrature Class Reference

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

Inheritance diagram for GaussianQuadrature:



### 7.257.1 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

## 7.258 GaussianStatistics Class Template Reference

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

### 7.258.1 Detailed Description

**template<class Stat> class QuantLib::GaussianStatistics< Stat >**

Statistics tool for gaussian-assumption risk measures.

It can calculate gaussian assumption risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the mean and variance provided by the template class

### Public Member Functions

- **GaussianStatistics** (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)*

### 7.258.2 Member Function Documentation

#### 7.258.2.1 **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  $x < 0$

**7.258.2.2 Real gaussianDownsideDeviation () const**

returns the downside deviation, defined as the square root of the downside variance.

**7.258.2.3 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)

**7.258.2.4 Real gaussianPercentile (Real percentile) const**

gaussian-assumption y-th percentile, defined as the value x such that

$$y = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp(-u^2/2) du$$

**7.258.2.5 Real gaussianTopPercentile (Real percentile) const**

**Precondition:**

percentile must be in range (0-100%) extremes excluded

**7.258.2.6 Real gaussianPotentialUpside (Real percentile) const**

gaussian-assumption Potential-Upside at a given percentile

**Precondition:**

percentile must be in range [90-100%)

**7.258.2.7 Real gaussianValueAtRisk (Real percentile) const**

gaussian-assumption Value-At-Risk at a given percentile

**Precondition:**

percentile must be in range [90-100%)

**7.258.2.8 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,

$$E[x \mid x < \text{VaR}(p)],$$

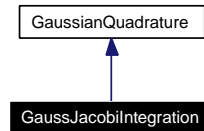
that is the average of observations below the given percentile  $p$ . Also known as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

## 7.259 GaussJacobiIntegration Class Reference

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

Inheritance diagram for GaussJacobiIntegration:



### 7.259.1 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

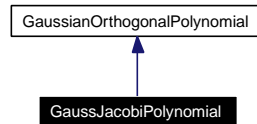
- `GaussJacobiIntegration` ([Size](#) n, [Real](#) alpha, [Real](#) beta)



## 7.260 GaussJacobiPolynomial Class Reference

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

Inheritance diagram for GaussJacobiPolynomial:



### 7.260.1 Detailed Description

Gauss-Jacobi polynomial.

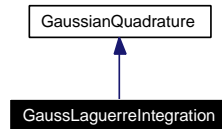
#### Public Member Functions

- `GaussJacobiPolynomial` ([Real](#) alpha, [Real](#) beta)
- [Real](#) `mu_0` () const
- [Real](#) `alpha` ([Size](#) i) const
- [Real](#) `beta` ([Size](#) i) const
- [Real](#) `w` ([Real](#) x) const

## 7.261 GaussLaguerreIntegration Class Reference

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

Inheritance diagram for GaussLaguerreIntegration:



### 7.261.1 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

$$s > -1$$

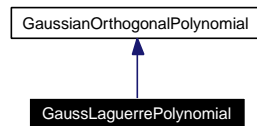
### Public Member Functions

- `GaussLaguerreIntegration` ([Size](#) n, [Real](#) s=0.0)

## 7.262 GaussLaguerrePolynomial Class Reference

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

Inheritance diagram for GaussLaguerrePolynomial:



### 7.262.1 Detailed Description

Gauss-Laguerre polynomial.

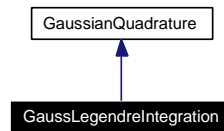
#### Public Member Functions

- `GaussLaguerrePolynomial` ([Real](#) s=0.0)
- [Real](#) `mu_0` () const
- [Real](#) `alpha` ([Size](#) i) const
- [Real](#) `beta` ([Size](#) i) const
- [Real](#) `w` ([Real](#) x) const

## 7.263 GaussLegendreIntegration Class Reference

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

Inheritance diagram for GaussLegendreIntegration:



### 7.263.1 Detailed Description

Gauss-Legendre integration.

This class performs a 1-dimensional Gauss-Legendre integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = 1$$

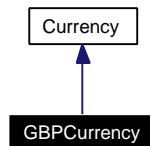
### Public Member Functions

- `GaussLegendreIntegration` ([Size](#) n)

## 7.264 GBPCurrency Class Reference

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

Inheritance diagram for GBPCurrency:



### 7.264.1 Detailed Description

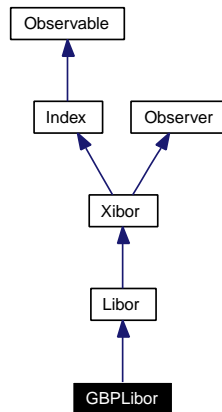
British pound sterling.

The ISO three-letter code is GBP; the numeric code is 826. It is divided into 100 pence.

## 7.265 GBPLibor Class Reference

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

Inheritance diagram for GBPLibor:



### 7.265.1 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** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual365Fixed](#)())

## 7.266 GeneralStatistics Class Reference

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

### 7.266.1 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.

Examples:

[DiscreteHedging.cpp](#).

### 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*

## 7.266.2 Member Function Documentation

### 7.266.2.1 `Real mean()` const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

### 7.266.2.2 `Real variance()` const

returns the variance, defined as

$$\sigma^2 = \frac{N}{N-1} \langle (x - \langle x \rangle)^2 \rangle.$$

### 7.266.2.3 `Real standardDeviation()` const

returns the standard deviation  $\sigma$ , defined as the square root of the variance.

### 7.266.2.4 `Real errorEstimate()` const

returns the error estimate on the mean value, defined as  $\epsilon = \sigma / \sqrt{N}$ .

### 7.266.2.5 `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.

### 7.266.2.6 `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.



**7.266.2.7 Real min () const**

returns the minimum sample value

**7.266.2.8 Real max () const**

returns the maximum sample value

**7.266.2.9 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.,

$$E[f | \mathcal{R}] = \frac{\sum_{x_i \in \mathcal{R}} f(x_i) w_i}{\sum_{x_i \in \mathcal{R}} w_i}.$$

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.

**7.266.2.10 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  $(0 - 1]$ .

**7.266.2.11 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  $(0 - 1]$ .

**7.266.2.12 void add (Real value, Real weight = 1.0)**

adds a datum to the set, possibly with a weight

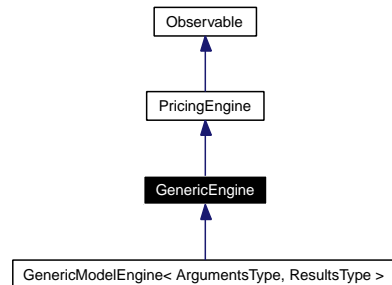
**Precondition:**

weights must be positive or null

## 7.267 GenericEngine Class Template Reference

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

Inheritance diagram for GenericEngine:



### 7.267.1 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`

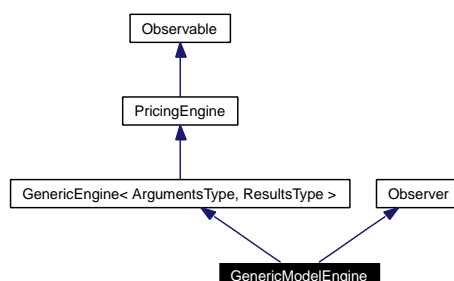
#### Protected Attributes

- `ArgumentsType arguments_`
- `ResultsType results_`

## 7.268 GenericModelEngine Class Template Reference

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

Inheritance diagram for GenericModelEngine:



### 7.268.1 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\_**

### 7.268.2 Member Function Documentation

#### 7.268.2.1 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< CapFloor::arguments, CapFloor::results >](#), [LatticeShortRateModelEngine< VanillaSwap::arguments, VanillaSwap::results >](#), and [LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >](#).

## 7.269 GenericRiskStatistics Class Template Reference

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

### 7.269.1 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 tool.

#### Todo

add historical annualized volatility

### 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

### 7.269.2 Member Function Documentation

#### 7.269.2.1 [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).

#### 7.269.2.2 [Real semiDeviation](#) () const

returns the semi deviation, defined as the square root of the semi variance.

**7.269.2.3 Real downsideVariance () const**

returns the variance of observations below 0.0,

$$\frac{N}{N-1} E[x^2 \mid x < 0].$$

**7.269.2.4 Real downsideDeviation () const**

returns the downside deviation, defined as the square root of the downside variance.

**7.269.2.5 Real regret (Real target) const**

returns the variance of observations below target,

$$\frac{N}{N-1} E[(x - t)^2 \mid x < t].$$

See Dembo and Freeman, "The Rules Of Risk", Wiley (2001).

**7.269.2.6 Real potentialUpside (Real centile) const**

potential upside (the reciprocal of VAR) at a given percentile

**Precondition:**

percentile must be in range [90-100%)

**7.269.2.7 Real valueAtRisk (Real centile) const**

value-at-risk at a given percentile

**Precondition:**

percentile must be in range [90-100%)

**7.269.2.8 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 know as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

### 7.269.2.9 **Real** shortfall (**Real** *target*) const

probability of missing the given target, defined as

$$E[\Theta \mid (-\infty, \infty)]$$

where

$$\Theta(x) = \begin{cases} 1 & x < t \\ 0 & x \geq t \end{cases}$$

### 7.269.2.10 **Real** averageShortfall (**Real** *target*) const

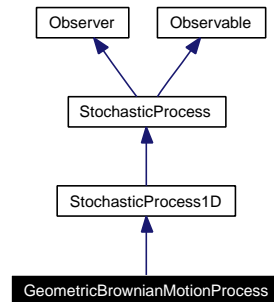
averaged shortfallness, defined as

$$E[t - x \mid x < t]$$

## 7.270 GeometricBrownianMotionProcess Class Reference

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

Inheritance diagram for GeometricBrownianMotionProcess:



### 7.270.1 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.  $\sigma(t, x_t)$*

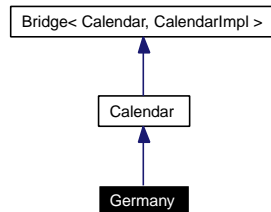
### Protected Attributes

- double **initialValue\_**
- double **mue\_**
- double **sigma\_**

## 7.271 Germany Class Reference

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

Inheritance diagram for Germany:



### 7.271.1 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.eurexchange.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)

## 7.271.2 Member Enumeration Documentation

### 7.271.2.1 enum [Market](#)

German calendars.

#### Enumerator:

*Settlement* generic settlement calendar

*FrankfurtStockExchange* Frankfurt stock-exchange.

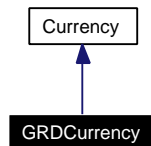
*Xetra* Xetra.

*Eurex* Eurex.

## 7.272 GRDCurrency Class Reference

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

Inheritance diagram for GRDCurrency:



### 7.272.1 Detailed Description

Greek drachma.

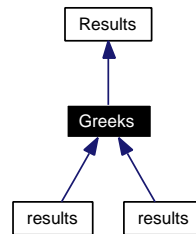
The ISO three-letter code was GRD; the numeric code was 300. It was divided in 100 lepta.

Obsoleted by the Euro since 2001.

## 7.273 Greeks Class Reference

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

Inheritance diagram for Greeks:



### 7.273.1 Detailed Description

additional option results

#### Public Member Functions

- `void reset ()`

#### Public Attributes

- [Real](#) delta
- [Real](#) gamma
- [Real](#) theta
- [Real](#) vega
- [Real](#) rho
- [Real](#) dividendRho

## 7.274 HaltonRsg Class Reference

```
#include <ql/RandomNumbers/haltonrsg.hpp>
```

### 7.274.1 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

## 7.275 Handle Class Template Reference

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

### 7.275.1 Detailed Description

**template<class Type> class QuantLib::Handle< Type >**

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 "Type" must inherit from [Observable](#)

#### Examples:

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

### Public Member Functions

- [Handle](#) (const boost::shared\_ptr< Type > &h=boost::shared\_ptr< Type >(), bool registerAsObserver=true)
- void [linkTo](#) (const boost::shared\_ptr< Type > &, bool registerAsObserver=true)
- const boost::shared\_ptr< Type > & [currentLink](#) () const  
*dereferencing*
- const boost::shared\_ptr< Type > & **operator** → () const
- bool [empty](#) () const  
*Checks if the contained shared pointer points to anything.*

### 7.275.2 Constructor & Destructor Documentation

7.275.2.1 [Handle](#) (const boost::shared\_ptr< Type > & h = boost::shared\_ptr< Type >(), bool *registerAsObserver* = true) [explicit]

#### Warning

see the documentation of the [Link](#) class for issues relatives to registerAsObserver.

### 7.275.3 Member Function Documentation

7.275.3.1 void [linkTo](#) (const boost::shared\_ptr< Type > &, bool *registerAsObserver* = true)

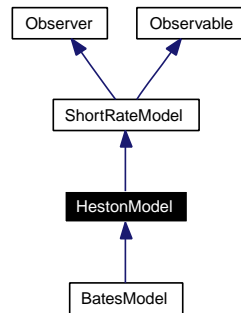
#### Warning

see the documentation of the [Link](#) class for issues relatives to registerAsObserver.

## 7.276 HestonModel Class Reference

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

Inheritance diagram for HestonModel:



### 7.276.1 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.

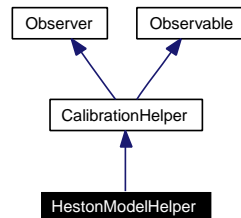
### Public Member Functions

- **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

## 7.277 HestonModelHelper Class Reference

```
#include <ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp>
```

Inheritance diagram for HestonModelHelper:



### 7.277.1 Detailed Description

calibration helper for Heston model

#### Public Member Functions

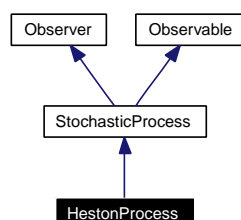
- **HestonModelHelper** (const [Period](#) &maturity, const [Calendar](#) &calendar, const [Real](#) s0, const [Real](#) strikePrice, const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [YieldTermStructure](#) > &dividendYield, bool 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



## 7.278 HestonProcess Class Reference

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

Inheritance diagram for HestonProcess:



### 7.278.1 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

## 7.278.2 Member Function Documentation

7.278.2.1 **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](#).

7.278.2.2 **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](#).

## 7.279 History Class Reference

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

### 7.279.1 Detailed Description

Container for historical data.

This class acts as a generic repository for a set of historical data. Single data can be accessed through their date, while sets of consecutive data can be accessed through iterators.

A history can contain null data, which can either be returned or skipped according to the chosen iterator type.

**Example:** [uses of history iterators](#)

### Public Types

- `typedef boost::filter_iterator< DataValidator, const\_iterator > const\_valid\_iterator`  
*bidirectional iterator on non-null history entries*
- `typedef std::vector< Real >::const_iterator const\_data\_iterator`  
*random access iterator on historical data*
- `typedef boost::filter_iterator< DataValidator, const\_data\_iterator > const\_valid\_data\_iterator`  
*bidirectional iterator on non-null historical data*

### Public Member Functions

- [History](#) ()
- `template<class Iterator> History (const Date &firstDate, const Date &lastDate, Iterator begin, Iterator end)`
- `History (const Date &firstDate, const std::vector< Real > &values)`
- `History (const Date &firstDate, const Date &lastDate, const std::vector< Real > &values)`
- `History (const std::vector< Date > &dates, const std::vector< Real > &values)`

### Inspectors

- `const Date & firstDate () const`  
*returns the first date for which a historical datum exists*
- `const Date & lastDate () const`  
*returns the last date for which a historical datum exists*
- `Size size () const`  
*returns the number of historical data including null ones*

### Historical data access

- [Real operator\[\]](#) (const [Date](#) &) const  
*returns the (possibly null) datum corresponding to the given date*

### Iterator access

Four different types of iterators are provided, namely, `const_iterator`, `const_valid_iterator`, `const_data_iterator`, and `const_valid_data_iterator`.

`const_iterator` and `const_valid_iterator` point to an `Entry` structure, the difference being that the latter only iterates over valid entries - i.e., entries whose data are not null. The same difference exists between `const_data_iterator` and `const_valid_data_iterator` which point directly to historical values without reference to the date they are associated to.

- `const_iterator begin ()` const
- `const_iterator end ()` const
- `const_iterator iterator (const Date &d)` const
- `const_valid_iterator vbegin ()` const
- `const_valid_iterator vend ()` const
- `const_valid_iterator valid_iterator (const Date &d)` const
- `const_data_iterator dbegin ()` const
- `const_data_iterator dend ()` const
- `const_data_iterator data_iterator (const Date &d)` const
- `const_valid_data_iterator vdbegin ()` const
- `const_valid_data_iterator vdend ()` const
- `const_valid_data_iterator valid_data_iterator (const Date &d)` const

## Classes

- class [const\\_iterator](#)  
*random access iterator on history entries*
- class [Entry](#)  
*single datum in history*

## 7.279.2 Constructor & Destructor Documentation

### 7.279.2.1 [History \(\)](#)

Default constructor

### 7.279.2.2 [History](#) (const [Date](#) & *firstDate*, const [Date](#) & *lastDate*, Iterator *begin*, Iterator *end*)

This constructor initializes the history with the given set of values, corresponding to the date range between *firstDate* and *lastDate* included.

#### Precondition:

*begin-end* must equal the number of days from *firstDate* to *lastDate* included.

### 7.279.2.3 **History** (const **Date** & *firstDate*, const **Date** & *lastDate*, const std::vector< **Real** > & *values*)

This constructor initializes the history with the given set of values, corresponding to the date range between *firstDate* and *lastDate* included.

**Precondition:**

The size of *values* must equal the number of days from *firstDate* to *lastDate* included.

### 7.279.2.4 **History** (const std::vector< **Date** > & *dates*, const std::vector< **Real** > & *values*)

This constructor initializes the history with the given set of values, corresponding each to the element with the same index in the given set of dates. The whole date range between *dates*[0] and *dates*[N-1] will be automatically filled by inserting null values where a date is missing from the given set.

**Precondition:**

*dates* must be sorted.

There can be no pairs (*dates*[i],*values*[i]) and (*dates*[j],*values*[j]) such that *dates*[i] == *dates*[j] && *values*[i] != *values*[j]. Pairs with *dates*[i] == *dates*[j] && *values*[i] == *values*[j] are allowed; the duplicated entries will be discarded.

The size of *values* must equal the number of days from *firstDate* to *lastDate* included.

## 7.280 History::const\_iterator Class Reference

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

### 7.280.1 Detailed Description

random access iterator on history entries

#### Public Member Functions

- const [Entry](#) & **dereference** () const
- bool **equal** (const [const\\_iterator](#) &i) const
- void **increment** ()
- void **decrement** ()
- void **advance** ([BigInteger](#) n)
- [BigInteger](#) **distance\_to** (const [const\\_iterator](#) &i) const

#### Friends

- class **History**

## 7.281 History::Entry Class Reference

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

### 7.281.1 Detailed Description

single datum in history

#### Public Member Functions

- const [Date](#) & **date** () const
- [Real](#) **value** () const

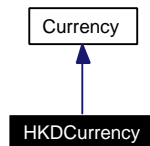
#### Friends

- class **const\_iterator**

## 7.282 HKDCurrency Class Reference

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

Inheritance diagram for HKDCurrency:



### 7.282.1 Detailed Description

Honk Kong dollar.

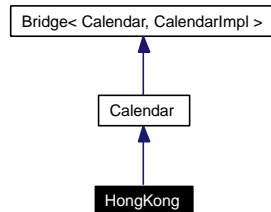
The ISO three-letter code is HKD; the numeric code is 344. It is divided in 100 cents.



## 7.283 HongKong Class Reference

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

Inheritance diagram for HongKong:



### 7.283.1 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](#))

## 7.283.2 Member Enumeration Documentation

### 7.283.2.1 enum [Market](#)

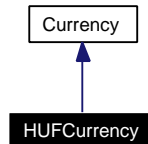
Enumerator:

*HKEx* Hong Kong stock exchange.

## 7.284 HUFCurrency Class Reference

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

Inheritance diagram for HUFCurrency:



### 7.284.1 Detailed Description

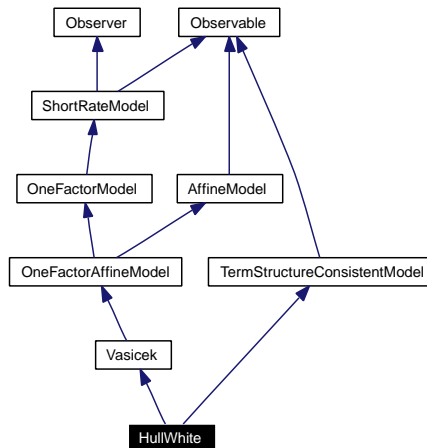
Hungarian forint.

The ISO three-letter code is HUF; the numeric code is 348. It has no subdivisions.

## 7.285 HullWhite Class Reference

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

Inheritance diagram for HullWhite:



### 7.285.1 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

## 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  $\varphi(t)$ .*

## 7.286 HullWhite::Dynamics Class Reference

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

### 7.286.1 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.

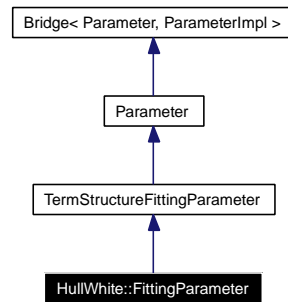
### Public Member Functions

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

## 7.287 HullWhite::FittingParameter Class Reference

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

Inheritance diagram for HullWhite::FittingParameter:



### 7.287.1 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  $t$ .

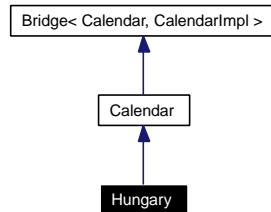
### Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a, [Real](#) sigma)

## 7.288 Hungary Class Reference

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

Inheritance diagram for Hungary:



### 7.288.1 Detailed Description

Hungarian calendar.

Holidays:

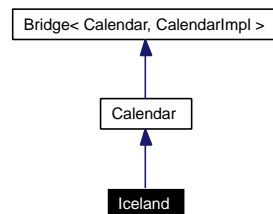
- Saturdays
- Sundays
- Easter Monday
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- National Day, March 15th
- Labour Day, May 1st
- Constitution Day, August 20th
- Republic Day, October 23rd
- All Saints Day, November 1st
- Christmas, December 25th
- 2nd Day of Christmas, December 26th



## 7.289 Iceland Class Reference

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

Inheritance diagram for Iceland:



### 7.289.1 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)

## 7.289.2 Member Enumeration Documentation

### 7.289.2.1 enum [Market](#)

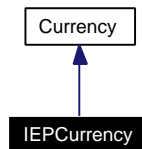
Enumerator:

*ICEX* [Iceland](#) stock exchange.

## 7.290 IEPCurrency Class Reference

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

Inheritance diagram for IEPCurrency:



### 7.290.1 Detailed Description

Irish punt.

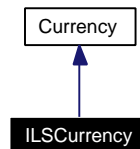
The ISO three-letter code was IEP; the numeric code was 372. It was divided in 100 pence.

Obsoleted by the Euro since 1999.

## 7.291 ILSCurrency Class Reference

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

Inheritance diagram for ILSCurrency:



### 7.291.1 Detailed Description

Israeli shekel.

The ISO three-letter code is ILS; the numeric code is 376. It is divided in 100 agorot.

## 7.292 IMM Struct Reference

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

### 7.292.1 Detailed Description

Main cycle of the International [Money](#) Market (a.k.a. [IMM](#)) Months.

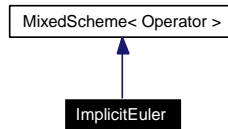
#### Public Types

- enum `Month` { `H` = 3, `M` = 6, `U` = 9, `Z` = 12 }

## 7.293 ImplicitEuler Class Template Reference

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

Inheritance diagram for ImplicitEuler:



### 7.293.1 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**

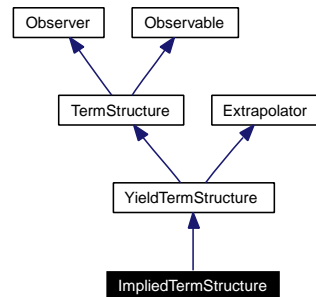
### Public Member Functions

- **ImplicitEuler** (const operator\_type &L, const bc\_set &bcs)

## 7.294 ImpliedTermStructure Class Reference

```
#include <ql/TermStructures/impliedtermstructure.hpp>
```

Inheritance diagram for ImpliedTermStructure:



### 7.294.1 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](#) &referenceDate)

#### 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 rates*

## Protected Member Functions

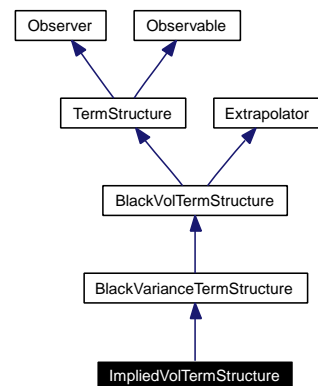
- [DiscountFactor](#) `discountImpl` ([Time](#)) const  
*returns the discount factor as seen from the evaluation date*



## 7.295 ImpliedVolTermStructure Class Reference

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

Inheritance diagram for ImpliedVolTermStructure:



### 7.295.1 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 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*

### Visitability

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

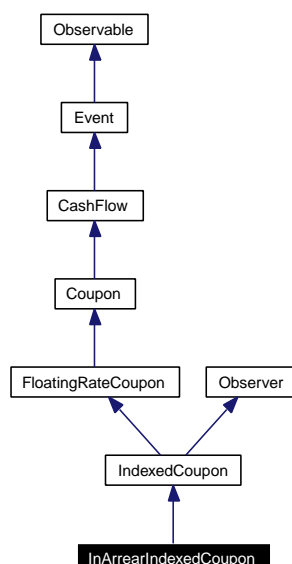
### Protected Member Functions

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

## 7.296 InArrearIndexedCoupon Class Reference

```
#include <ql/CashFlows/inarrearindexedcoupon.hpp>
```

Inheritance diagram for InArrearIndexedCoupon:



### 7.296.1 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** (*Real* nominal, const *Date* &paymentDate, const boost::shared\_ptr< *Xibor* > &index, const *Date* &startDate, const *Date* &endDate, *Integer* fixingDays, *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 convexityAdjustment](#) ([Rate](#) fixing) const  
*convexity adjustment for the given index fixing*

#### Protected Attributes

- boost::shared\_ptr< [Xibor](#) > **xibor\_**
- [Handle](#)< [CapletVolatilityStructure](#) > **capletVolatility\_**

## 7.297 IncrementalStatistics Class Reference

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

### 7.297.1 Detailed Description

Statistics tool based on incremental accumulation.

It can accumulate a set of data and return statistics (e.g: mean, variance, skewness, kurtosis, error estimation, etc.)

#### Warning

high moments are numerically unstable for high average/standardDeviation ratios.

### Public Member Functions

#### Inspectors

- [Size samples](#) () const  
*number of samples collected*
- [Real weightSum](#) () const  
*sum of data weights*
- [Real mean](#) () const
- [Real variance](#) () const
- [Real standardDeviation](#) () const
- [Real downsideVariance](#) () const
- [Real downsideDeviation](#) () const
- [Real errorEstimate](#) () const
- [Real skewness](#) () const
- [Real kurtosis](#) () const
- [Real min](#) () const
- [Real max](#) () 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\_

## 7.297.2 Member Function Documentation

### 7.297.2.1 [Real](#) mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

### 7.297.2.2 [Real](#) variance () const

returns the variance, defined as

$$\frac{N}{N-1} \langle (x - \langle x \rangle)^2 \rangle.$$

### 7.297.2.3 [Real](#) standardDeviation () const

returns the standard deviation  $\sigma$ , defined as the square root of the variance.

### 7.297.2.4 [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  $x < 0$

### 7.297.2.5 [Real](#) downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

**7.297.2.6 Real errorEstimate () const**

returns the error estimate  $\epsilon$ , defined as the square root of the ratio of the variance to the number of samples.

**7.297.2.7 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.

**7.297.2.8 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.

**7.297.2.9 Real min () const**

returns the minimum sample value

**7.297.2.10 Real max () const**

returns the maximum sample value

**7.297.2.11 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

**7.297.2.12 void addSequence (DataIterator begin, DataIterator end, WeightIterator wbegin)**

adds a sequence of data to the set, each with its weight

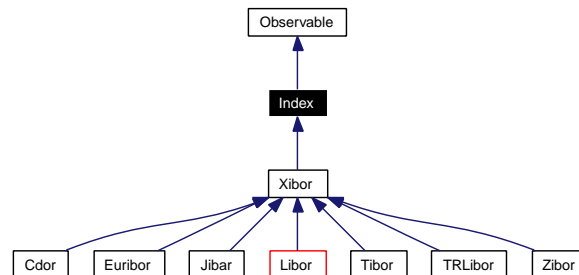
**Precondition:**

weights must be positive or null

## 7.298 Index Class Reference

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

Inheritance diagram for Index:



### 7.298.1 Detailed Description

purely virtual base class for indexes

#### Public Member Functions

- virtual `std::string name () const =0`  
*Returns the name of the index.*
- virtual `Rate fixing (const Date &fixingDate) const =0`  
*returns the fixing at the given date*

### 7.298.2 Member Function Documentation

#### 7.298.2.1 virtual `std::string name () const` [pure virtual]

Returns the name of the index.

##### Warning

This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implemented in [Xibor](#).

#### 7.298.2.2 virtual `Rate fixing (const Date &fixingDate) const` [pure virtual]

returns the fixing at the given date

##### Note:

any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

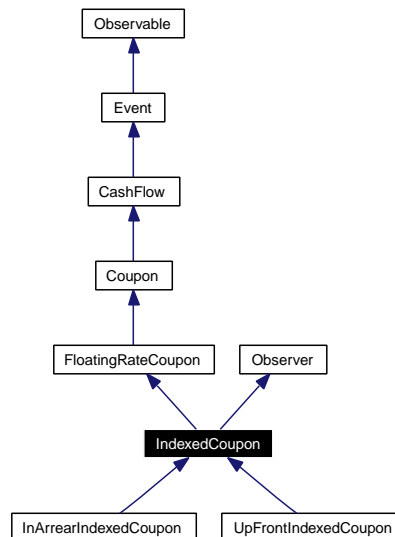
Implemented in [Xibor](#).



## 7.299 IndexedCoupon Class Reference

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

Inheritance diagram for IndexedCoupon:



### 7.299.1 Detailed Description

Base indexed coupon class.

#### Warning

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

### Public Member Functions

- **IndexedCoupon** (*Real* nominal, const *Date* &paymentDate, const boost::shared\_ptr< *Index* > &index, const *Date* &startDate, const *Date* &endDate, *Integer* fixingDays, *Spread* spread=0.0, const *Date* &refPeriodStart=*Date*(), const *Date* &refPeriodEnd=*Date*(), const *DayCounter* &dayCounter=*DayCounter*())

#### CashFlow interface

- *Real* amount () const  
*returns the amount of the cash flow*

#### Coupon interface

- *DayCounter* dayCounter () const  
*day counter for accrual calculation*

### FloatingRateCoupon interface

- [Rate indexFixing](#) () const  
*fixing of the underlying index*

### Inspectors

- const boost::shared\_ptr< [Index](#) > & [index](#) () const

### Observer interface

- void [update](#) ()

### Visitability

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

## 7.299.2 Member Function Documentation

### 7.299.2.1 [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](#).

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

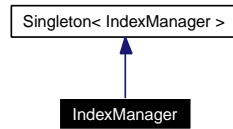
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.300 IndexManager Class Reference

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

Inheritance diagram for IndexManager:



### 7.300.1 Detailed Description

global repository for past index fixings

#### Public Member Functions

- void **setHistory** (const std::string &name, const [History](#) &)
- const [History](#) & **getHistory** (const std::string &name) const
- bool **hasHistory** (const std::string &name) const
- std::vector< std::string > **histories** () const

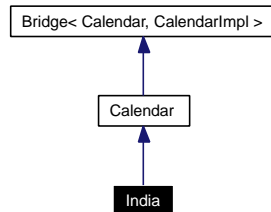
#### Friends

- class **Singleton< IndexManager >**

## 7.301 India Class Reference

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

Inheritance diagram for India:



### 7.301.1 Detailed Description

Indian calendars.

Holidays for the National [Stock](http://www.nse-india.com/) Exchange (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)

## 7.301.2 Member Enumeration Documentation

### 7.301.2.1 enum [Market](#)

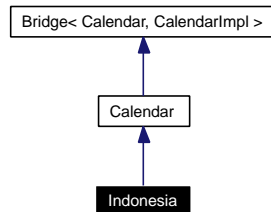
#### Enumerator:

[NSE](#) National [Stock](#) Exchange.

## 7.302 Indonesia Class Reference

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

Inheritance diagram for Indonesia:



### 7.302.1 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)

## 7.302.2 Member Enumeration Documentation

### 7.302.2.1 enum [Market](#)

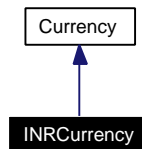
Enumerator:

*BEJ* Jakarta stock exchange.

## 7.303 INRCurrency Class Reference

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

Inheritance diagram for INRCurrency:



### 7.303.1 Detailed Description

Indian rupee.

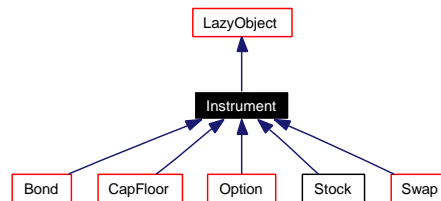
The ISO three-letter code is INR; the numeric code is 356. It is divided in 100 paise.



## 7.304 Instrument Class Reference

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

Inheritance diagram for Instrument:



### 7.304.1 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_`

## 7.304.2 Member Function Documentation

### 7.304.2.1 `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.

### 7.304.2.2 `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](#), [BarrierOption](#), [BasketOption](#), [CapFloor](#), [CliquetOption](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoForwardVanillaOption](#), [QuantoVanillaOption](#), [VanillaSwap](#), and [Swaption](#).

### 7.304.2.3 `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 [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), and [VanillaSwap](#).

### 7.304.2.4 `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](#).

#### 7.304.2.5 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](#), and [Swap](#).

#### 7.304.2.6 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](#), [Stock](#), and [Swap](#).

## 7.305 IntegralEngine Class Reference

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

### 7.305.1 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

## 7.306 InterestRate Class Reference

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

### 7.306.1 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

#### constructors

- [InterestRate](#) ()  
*Default constructor returning a null interest rate.*
- [InterestRate](#) ([Rate](#) r, const [DayCounter](#) &dc, Compounding comp, [Frequency](#) freq=[Annual](#))  
*Standard constructor.*

#### 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 &)`

## 7.306.2 Member Function Documentation

### 7.306.2.1 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.

### 7.306.2.2 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.

7.306.2.3 **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.

7.306.2.4 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.

7.306.2.5 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.

7.306.2.6 **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.

7.306.2.7 **InterestRate** equivalentRate (**Date** d1, **Date** d2, const **DayCounter** & resultDC, Compounding comp, **Frequency** freq = Annual) const

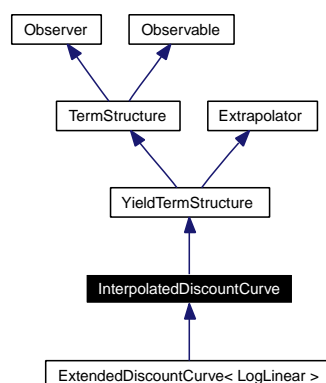
equivalent rate for a compounding period between two dates

The resulting rate is calculated taking the required day-counting rule into account.

## 7.307 InterpolatedDiscountCurve Class Template Reference

```
#include <ql/TermStructures/discountcurve.hpp>
```

Inheritance diagram for InterpolatedDiscountCurve:



### 7.307.1 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 rates*
- [Time](#) **maxTime** () const  
*the latest time for which the curve can return rates*
- const std::vector< [Time](#) > & **times** () const
- const std::vector< [Date](#) > & **dates** () const
- const std::vector< [DiscountFactor](#) > & **discounts** () const

### Protected 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*

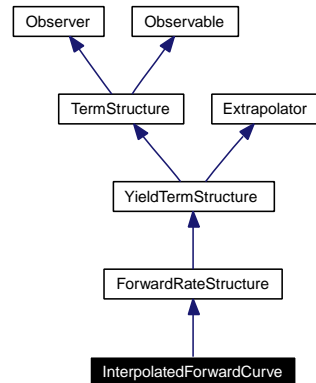
## Protected Attributes

- [DayCounter](#) **dayCounter\_**
- std::vector< [Date](#) > **dates\_**
- std::vector< [Time](#) > **times\_**
- std::vector< [DiscountFactor](#) > **data\_**
- [Interpolation](#) **interpolation\_**
- Interpolator **interpolator\_**

## 7.308 InterpolatedForwardCurve Class Template Reference

```
#include <ql/TermStructures/forwardcurve.hpp>
```

Inheritance diagram for InterpolatedForwardCurve:



### 7.308.1 Detailed Description

```
template<class Interpolator> class QuantLib::InterpolatedForwardCurve< Interpolator >
```

Term structure based on interpolation of forward rates.

#### Inspectors

- [DayCounter dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Date maxDate](#) () const  
*the latest date for which the curve can return rates*
- [Time maxTime](#) () const  
*the latest time for which the curve can return rates*
- const std::vector< [Time](#) > & [times](#) () const
- const std::vector< [Date](#) > & [dates](#) () const
- [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
- [DayCounter dayCounter\\_](#)

- `std::vector< Date > dates_`
- `std::vector< Time > times_`
- `std::vector< Rate > data_`
- `Interpolation interpolation_`
- `Interpolator interpolator_`

## Public Member Functions

- **InterpolatedForwardCurve** (`const std::vector< Date > &dates`, `const std::vector< Rate > &forwards`, `const DayCounter &dayCounter`, `const Interpolator &interpolator=Interpolator()`)

## 7.308.2 Member Function Documentation

### 7.308.2.1 [Rate](#) zeroYieldImpl ([Time](#) *t*) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

#### [Warning](#)

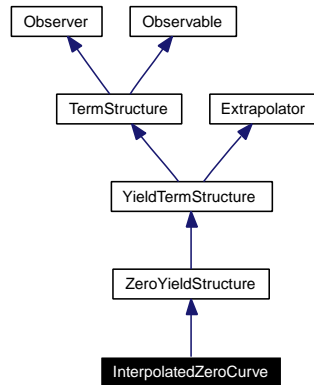
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Reimplemented from [ForwardRateStructure](#).

## 7.309 InterpolatedZeroCurve Class Template Reference

```
#include <ql/TermStructures/zerocurve.hpp>
```

Inheritance diagram for InterpolatedZeroCurve:



### 7.309.1 Detailed Description

```
template<class Interpolator> class QuantLib::InterpolatedZeroCurve< Interpolator >
```

Term structure based on interpolation of zero yields.

#### Inspectors

- [DayCounter dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Date maxDate](#) () const  
*the latest date for which the curve can return rates*
- [Time maxTime](#) () const  
*the latest time for which the curve can return rates*
- const std::vector< [Time](#) > & [times](#) () const
- const std::vector< [Date](#) > & [dates](#) () const
- [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*
- [DayCounter dayCounter\\_](#)
- std::vector< [Date](#) > [dates\\_](#)

- `std::vector< Time > times_`
- `std::vector< Rate > data_`
- `Interpolation interpolation_`
- `Interpolator interpolator_`

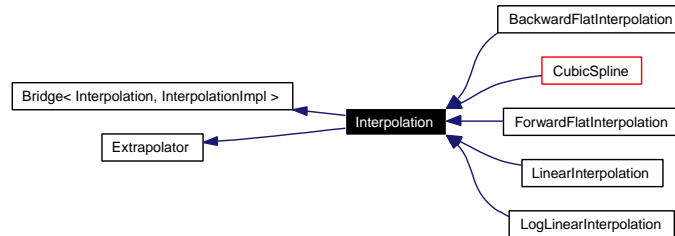
## Public Member Functions

- **InterpolatedZeroCurve** (const `std::vector< Date > &dates`, const `std::vector< Rate > &yields`, const `DayCounter &dayCounter`, const `Interpolator &interpolator=Interpolator()`)

## 7.310 Interpolation Class Reference

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

Inheritance diagram for Interpolation:



### 7.310.1 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

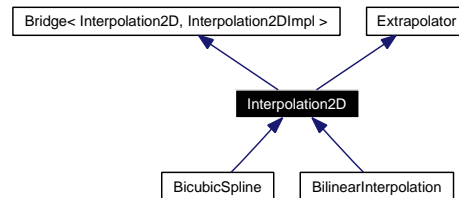
#### Classes

- class [templateImpl](#)  
*basic template implementation*

## 7.311 Interpolation2D Class Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2D:



### 7.311.1 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
- [Real](#) `yMin` () const
- [Real](#) `yMax` () const
- bool `isInRange` ([Real](#)  $x$ , [Real](#)  $y$ ) const
- void `update` ()

### Protected Member Functions

- void `checkRange` ([Real](#)  $x$ , [Real](#)  $y$ , bool `extrapolate`) const

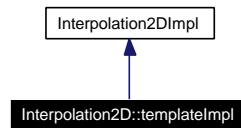
### Classes

- class [templateImpl](#)  
*basic template implementation*

## 7.312 Interpolation2D::templateImpl Class Template Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2D::templateImpl:



### 7.312.1 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
- [Real](#) **yMin** () const
- [Real](#) **yMax** () const
- [bool](#) **isInRange** ([Real](#) x, [Real](#) y) const

#### Protected Member Functions

- [Size](#) **locateX** ([Real](#) x) const
- [Size](#) **locateY** ([Real](#) y) const

#### Protected Attributes

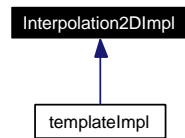
- I1 **xBegin\_**
- I1 **xEnd\_**
- I2 **yBegin\_**
- I2 **yEnd\_**
- const M & **zData\_**



## 7.313 Interpolation2DImpl Class Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2DImpl:



### 7.313.1 Detailed Description

abstract base class for 2-D interpolation implementations

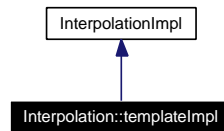
#### Public Member Functions

- virtual void **calculate** ()=0
- virtual [Real](#) **xMin** () const =0
- virtual [Real](#) **xMax** () const =0
- virtual [Real](#) **yMin** () const =0
- virtual [Real](#) **yMax** () const =0
- virtual bool **isInRange** ([Real](#) x, [Real](#) y) const =0
- virtual [Real](#) **value** ([Real](#) x, [Real](#) y) const =0

## 7.314 Interpolation::templateImpl Class Template Reference

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

Inheritance diagram for Interpolation::templateImpl:



### 7.314.1 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

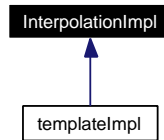
#### Protected Attributes

- I1 **xBegin\_**
- I1 **xEnd\_**
- I2 **yBegin\_**

## 7.315 InterpolationImpl Class Reference

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

Inheritance diagram for InterpolationImpl:



### 7.315.1 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

## 7.316 InverseCumulativeNormal Class Reference

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

### 7.316.1 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

## 7.317 InverseCumulativePoisson Class Reference

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

### 7.317.1 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

## 7.318 InverseCumulativeRng Class Template Reference

```
#include <ql/RandomNumbers/inversecumulativrng.hpp>
```

### 7.318.1 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*

## 7.319 InverseCumulativeRsg Class Template Reference

```
#include <ql/RandomNumbers/inversecumulativrsg.hpp>
```

### 7.319.1 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;
```

### Public Types

- typedef [Sample< Array >](#) **sample\_type**

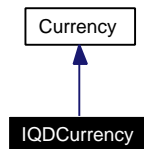
### Public Member Functions

- **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

## 7.320 IQDCurrency Class Reference

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

Inheritance diagram for IQDCurrency:



### 7.320.1 Detailed Description

Iraqi dinar.

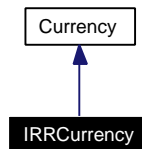
The ISO three-letter code is IQD; the numeric code is 368. It is divided in 1000 fils.



## 7.321 IRRCurrency Class Reference

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

Inheritance diagram for IRRCurrency:



### 7.321.1 Detailed Description

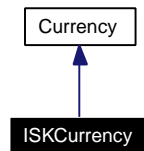
Iranian rial.

The ISO three-letter code is IRR; the numeric code is 364. It has no subdivisions.

## 7.322 ISKCurrency Class Reference

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

Inheritance diagram for ISKCurrency:



### 7.322.1 Detailed Description

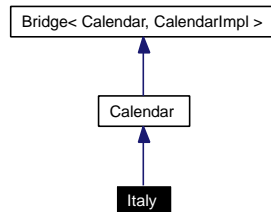
[Iceland](#) krona.

The ISO three-letter code is ISK; the numeric code is 352. It is divided in 100 aurar.

## 7.323 Italy Class Reference

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

Inheritance diagram for Italy:



### 7.323.1 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)

## 7.323.2 Member Enumeration Documentation

### 7.323.2.1 enum [Market](#)

Italian calendars.

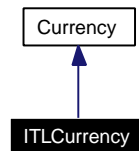
#### Enumerator:

- Settlement* generic settlement calendar
- Exchange* Milan stock-exchange calendar.

## 7.324 ITLCurrency Class Reference

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

Inheritance diagram for ITLCurrency:



### 7.324.1 Detailed Description

Italian lira.

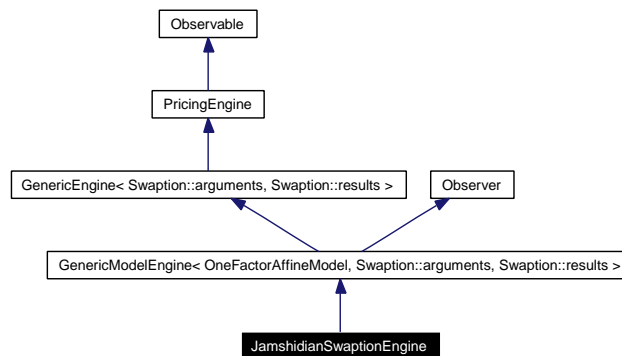
The ISO three-letter code was ITL; the numeric code was 380. It had no subdivisions.

Obsoleted by the Euro since 1999.

## 7.325 JamshidianSwaptionEngine Class Reference

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

Inheritance diagram for JamshidianSwaptionEngine:



### 7.325.1 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

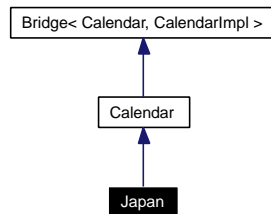
### Friends

- class **rStarFinder**

## 7.326 Japan Class Reference

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

Inheritance diagram for Japan:



### 7.326.1 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

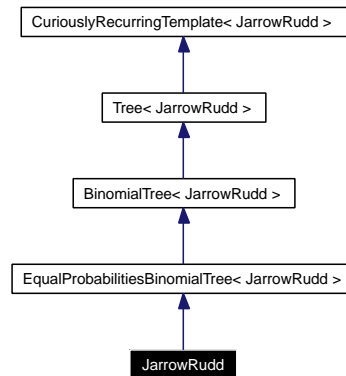
Holidays falling on a Sunday are observed on the Monday following except for the bank holidays associated with the new year.



## 7.327 JarrowRudd Class Reference

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

Inheritance diagram for JarrowRudd:



### 7.327.1 Detailed Description

Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

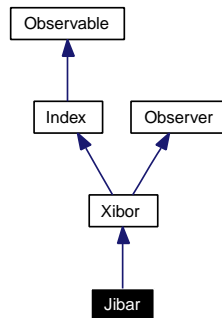
#### Public Member Functions

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

## 7.328 Jibar Class Reference

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

Inheritance diagram for Jibar:



### 7.328.1 Detailed Description

JIBAR rate

Johannesburg Interbank Agreed Rate

#### Todo

check settlement days and day-count convention.

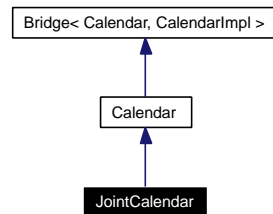
### Public Member Functions

- **Jibar**([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [DayCounter](#) &dc=[Actual365Fixed](#)())

## 7.329 JointCalendar Class Reference

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

Inheritance diagram for JointCalendar:



### 7.329.1 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.

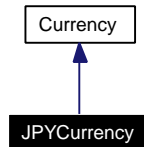
### Public Member Functions

- **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)

## 7.330 JPYCurrency Class Reference

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

Inheritance diagram for JPYCurrency:



### 7.330.1 Detailed Description

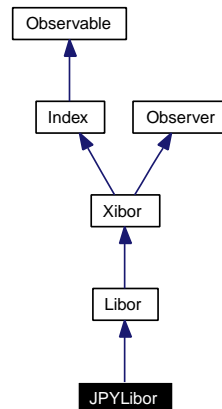
Japanese yen.

The ISO three-letter code is JPY; the numeric code is 392. It is divided into 100 sen.

## 7.331 JPYLibor Class Reference

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

Inheritance diagram for JPYLibor:



### 7.331.1 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.

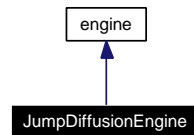
### Public Member Functions

- **JPYLibor** (*Integer* n, *TimeUnit* units, const *Handle*< *YieldTermStructure* > &h, const *Day-Counter* &dc=*Actual360*())

## 7.332 JumpDiffusionEngine Class Reference

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

Inheritance diagram for JumpDiffusionEngine:



### 7.332.1 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.

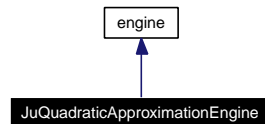
### Public Member Functions

- **JumpDiffusionEngine** (const boost::shared\_ptr< [VanillaOption::engine](#) > &, [Real](#) relative-Accuracy\_=1e-4, Size maxIterations=100)
- void **calculate** () const

## 7.333 JuQuadraticApproximationEngine Class Reference

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

Inheritance diagram for JuQuadraticApproximationEngine:



### 7.333.1 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`

## 7.334 KnuthUniformRng Class Reference

```
#include <ql/RandomNumbers/knuthuniformrng.hpp>
```

### 7.334.1 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

### 7.334.2 Constructor & Destructor Documentation

#### 7.334.2.1 [KnuthUniformRng](#) (long *seed* = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

### 7.334.3 Member Function Documentation

#### 7.334.3.1 [KnuthUniformRng::sample\\_type](#) `next ()` const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)



## 7.335 KronrodIntegral Class Reference

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

### 7.335.1 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.

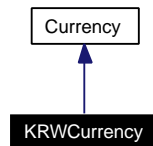
### Public Member Functions

- **KronrodIntegral** ([Real](#) tolerance, [Size](#) maxFunctionEvaluations=[Null](#)< [Size](#) >())
- `template<class F> Real operator()` (const F &f, [Real](#) a, [Real](#) b) const
- [Size](#) `functionEvaluations` ()

## 7.336 KRWCurrency Class Reference

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

Inheritance diagram for KRWCurrency:



### 7.336.1 Detailed Description

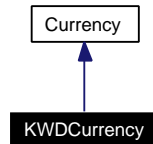
South-Korean won.

The ISO three-letter code is KRW; the numeric code is 410. It is divided in 100 chon.

## 7.337 KWDCurrency Class Reference

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

Inheritance diagram for KWDCurrency:



### 7.337.1 Detailed Description

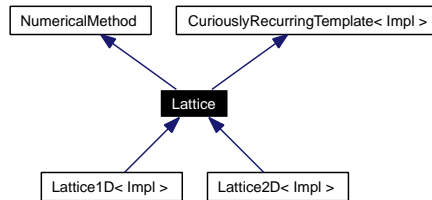
Kuwaiti dinar.

The ISO three-letter code is KWD; the numeric code is 414. It is divided in 1000 fils.

## 7.338 Lattice Class Template Reference

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

Inheritance diagram for Lattice:



### 7.338.1 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_`

### 7.338.2 Member Function Documentation

#### 7.338.2.1 void `rollback` ([DiscretizedAsset](#) &, [Time](#) to) const [virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implements [NumericalMethod](#).

Reimplemented in [TsiveriotisFernandesLattice](#).

#### 7.338.2.2 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();
```

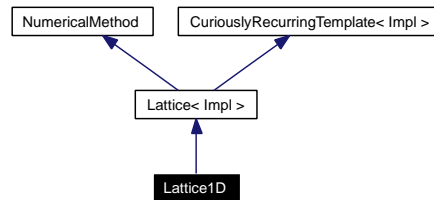
Implements [NumericalMethod](#).

Reimplemented in [TsiveriotisFernandesLattice](#).

## 7.339 Lattice1D Class Template Reference

```
#include <ql/Lattices/lattice1d.hpp>
```

Inheritance diagram for Lattice1D:



### 7.339.1 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;
```

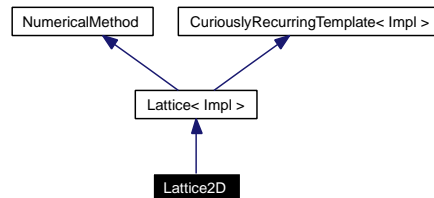
#### Public Member Functions

- **Lattice1D** (const [TimeGrid](#) &timeGrid, [Size](#) n)
- **Disposable**< [Array](#) > **grid** ([Time](#) t) const
- **Real** **underlying** ([Size](#) i, [Size](#) index) const

## 7.340 Lattice2D Class Template Reference

```
#include <ql/Lattices/lattice2d.hpp>
```

Inheritance diagram for Lattice2D:



### 7.340.1 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

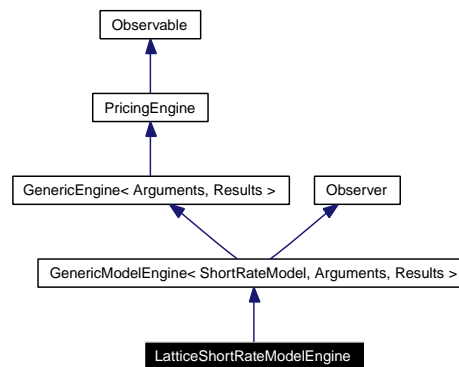
#### Protected Attributes

- boost::shared\_ptr< T > **tree1\_**
- boost::shared\_ptr< T > **tree2\_**

## 7.341 LatticeShortRateModelEngine Class Template Reference

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

Inheritance diagram for LatticeShortRateModelEngine:



### 7.341.1 Detailed Description

**template<class Arguments, class Results> class QuantLib::LatticeShortRateModelEngine< Arguments, Results >**

Engine for a short-rate model specialized on a lattice.

Derived engines only need to implement the `calculate()` method

### Public Member Functions

- **LatticeShortRateModelEngine** (const 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\_

### 7.341.2 Member Function Documentation

#### 7.341.2.1 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.

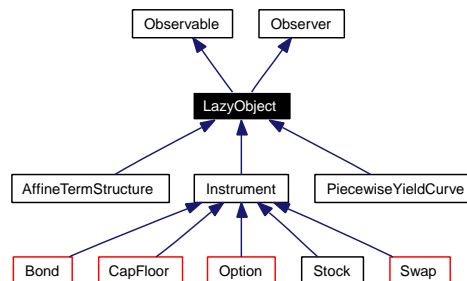
Reimplemented from [GenericModelEngine< ShortRateModel, Arguments, Results >](#).



## 7.342 LazyObject Class Reference

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

Inheritance diagram for LazyObject:



### 7.342.1 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_`

### 7.342.2 Member Function Documentation

#### 7.342.2.1 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 [AffineTermStructure](#), and [PiecewiseYieldCurve](#).

#### 7.342.2.2 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.

#### 7.342.2.3 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.

#### 7.342.2.4 void unfreeze ()

This method reverts the effect of the **freeze** method, thus re-enabling recalculations.

#### 7.342.2.5 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](#).

#### 7.342.2.6 virtual void performCalculations () const [protected, pure virtual]

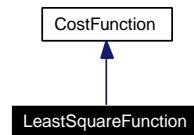
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implemented in [Instrument](#), [Bond](#), [Stock](#), and [Swap](#).

## 7.343 LeastSquareFunction Class Reference

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

Inheritance diagram for LeastSquareFunction:



### 7.343.1 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*

## 7.344 LeastSquareProblem Class Reference

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

### 7.344.1 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

### 7.344.2 Member Function Documentation

**7.344.2.1** 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

## 7.345 LecuyerUniformRng Class Reference

```
#include <ql/RandomNumbers/lecuyeruniformrng.hpp>
```

### 7.345.1 Detailed Description

Uniform random number generator.

Random number generator of L'Ecuyer with added Bays-Durham shuffle (know as ran2 in Numerical recipes)

For more details see Section 7.1 of Numerical Recipes in C, 2nd Edition, Cambridge University Press (available at <http://www.nr.com/>)

### Public Types

- typedef [Sample< Real >](#) `sample_type`

### Public Member Functions

- [LecuyerUniformRng](#) (long seed=0)
- [sample\\_type next](#) () const

### 7.345.2 Constructor & Destructor Documentation

#### 7.345.2.1 [LecuyerUniformRng](#) (long *seed* = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

### 7.345.3 Member Function Documentation

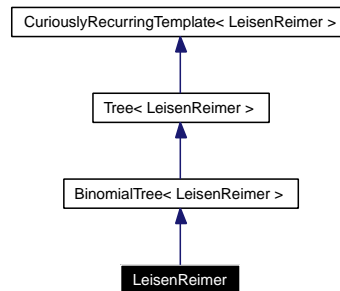
#### 7.345.3.1 [sample\\_type next](#) () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

## 7.346 LeisenReimer Class Reference

#include <ql/Lattices/binomialtree.hpp>

Inheritance diagram for LeisenReimer:



### 7.346.1 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

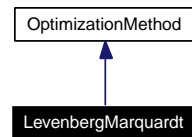
#### Protected Attributes

- [Real](#) up\_
- [Real](#) down\_
- [Real](#) pu\_
- [Real](#) pd\_

## 7.347 LevenbergMarquardt Class Reference

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

Inheritance diagram for LevenbergMarquardt:



### 7.347.1 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=200)
- 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)

### 7.347.2 Constructor & Destructor Documentation

7.347.2.1 [LevenbergMarquardt](#) ([Real](#) epsfcn = 1e-8, [Real](#) ftol = 1e-8, [Real](#) xtol = 1e-8, [Real](#) gtol = 1e-8, [Size](#) maxfev = 200)

Constructor taking as input the characteristic length and tolerance

## 7.348 LexicographicalView Class Template Reference

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

### 7.348.1 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)



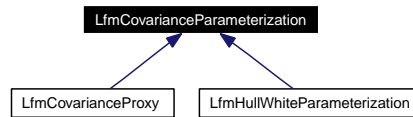
### Inspectors

- [Size xSize \(\)](#) const  
*dimension of the array along x*
- [Size ySize \(\)](#) const  
*dimension of the array along y*

## 7.349 LfmCovarianceParameterization Class Reference

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

Inheritance diagram for LfmCovarianceParameterization:



### 7.349.1 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

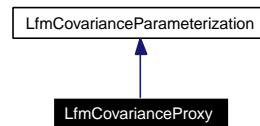
### Protected Attributes

- const [Size](#) size\_
- const [Size](#) factors\_

## 7.350 LfmCovarianceProxy Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lfmcovarproxy.hpp>
```

Inheritance diagram for LfmCovarianceProxy:



### 7.350.1 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\_**

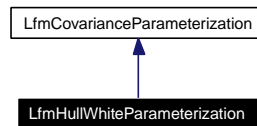
#### Friends

- class **Var\_Helper**

## 7.351 LfmHullWhiteParameterization Class Reference

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

Inheritance diagram for LfmHullWhiteParameterization:



### 7.351.1 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>)

#### 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

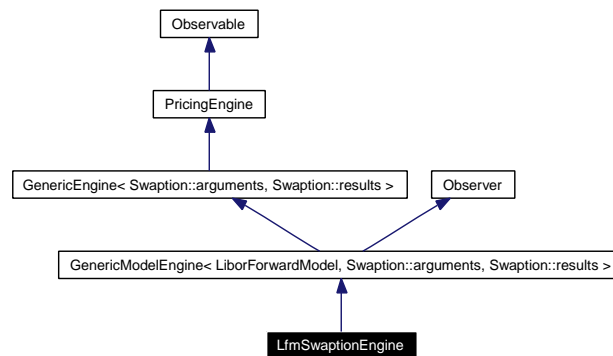
### Protected Attributes

- [Matrix](#) **diffusion\_**
- [Matrix](#) **covariance\_**
- std::vector< [Time](#) > **fixingTimes\_**

## 7.352 LfmSwaptionEngine Class Reference

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

Inheritance diagram for LfmSwaptionEngine:



### 7.352.1 Detailed Description

libor forward model swaption engine based on black formula

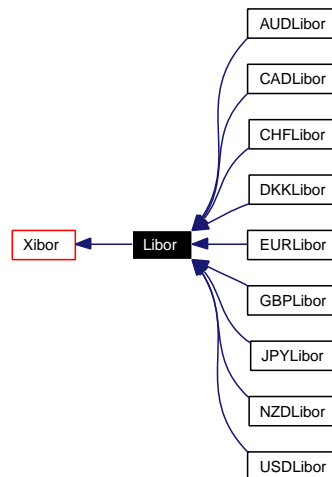
#### Public Member Functions

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

## 7.353 Libor Class Reference

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

Inheritance diagram for Libor:



### 7.353.1 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)

#### Date calculations

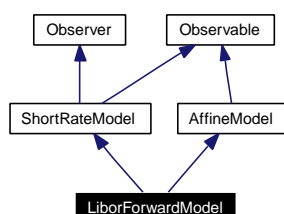
see <http://www.bba.org.uk/bba/jsp/polopoly.jsp?d=225&a=1412>

- **Date valueDate** (const Date &fixingDate) const
- **Date maturityDate** (const Date &valueDate) const

## 7.354 LiborForwardModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp>
```

Inheritance diagram for LiborForwardModel:



### 7.354.1 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)
- boost::shared\_ptr< **NumericalMethod** > **tree** (const **TimeGrid** &) const

### Protected Member Functions

- **Disposable**< **Array** > **w\_0** (**Size** alpha, **Size** beta) const

## Protected Attributes

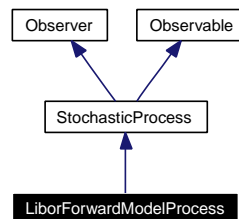
- `std::vector< Real > f_`
- `std::vector< Time > accrualPeriod_`
- `const boost::shared_ptr< LfmCovarianceProxy > covarProxy_`
- `const boost::shared_ptr< LiborForwardModelProcess > process_`
- `boost::shared_ptr< SwaptionVolatilityMatrix > swaptionVola`



## 7.355 LiborForwardModelProcess Class Reference

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

Inheritance diagram for LiborForwardModelProcess:



### 7.355.1 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) (<<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

## 7.355.2 Member Function Documentation

### 7.355.2.1 [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](#).

### 7.355.2.2 [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](#).

### 7.355.2.3 [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](#).

## 7.356 Linear Class Reference

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

### 7.356.1 Detailed Description

[Linear](#) interpolation factory and traits.

#### Public Types

- enum { **global** = 0 }

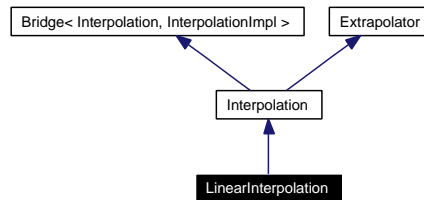
#### Public Member Functions

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

## 7.357 LinearInterpolation Class Reference

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

Inheritance diagram for LinearInterpolation:



### 7.357.1 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)

### 7.357.2 Constructor & Destructor Documentation

#### 7.357.2.1 [LinearInterpolation](#) (const I1 & xBegin, const I1 & xEnd, const I2 & yBegin)

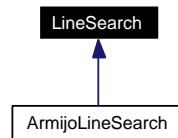
**Precondition:**

the  $x$  values must be sorted.

## 7.358 LineSearch Class Reference

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

Inheritance diagram for LineSearch:



### 7.358.1 Detailed Description

Base class for line search.

#### Public Member Functions

- [LineSearch](#) ([Real](#) eps=1e-8)  
*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

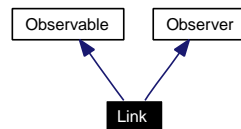
- [Array](#) [xtd\\_](#)  
*new x and its gradient*
- [Array](#) [gradient\\_](#)

- [Real qt\\_](#)  
*cost function value and gradient norm corresponding to xtd\_*
- [Real qpt\\_](#)
- [bool succeed\\_](#)  
*flag to know if linesearch succeed*

## 7.359 Link Class Template Reference

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

Inheritance diagram for Link:



### 7.359.1 Detailed Description

```
template<class Type> class QuantLib::Link< Type >
```

Relinkable access to a shared pointer.

#### Precondition:

Class "Type" must inherit from [Observable](#)

### Public Member Functions

- [Link](#) (const boost::shared\_ptr< Type > &h=boost::shared\_ptr< Type >(), bool registerAsObserver=true)
- void [linkTo](#) (const boost::shared\_ptr< Type > &, bool registerAsObserver=true)
- bool [empty](#) () const  
*Checks if the contained shared pointer points to anything.*
- const boost::shared\_ptr< Type > & [currentLink](#) () const  
*Returns the contained shared pointer.*
- void [update](#) ()  
*[Observer](#) interface.*

### 7.359.2 Constructor & Destructor Documentation

7.359.2.1 [Link](#) (const boost::shared\_ptr< Type > &h = boost::shared\_ptr< Type >(), bool registerAsObserver = true) [explicit]

#### Warning

see the documentation of the [linkTo\(\)](#) method for issues relatives to registerAsObserver.

### 7.359.3 Member Function Documentation

#### 7.359.3.1 `void linkTo (const boost::shared_ptr< Type > &, bool registerAsObserver = true)`

##### Warning

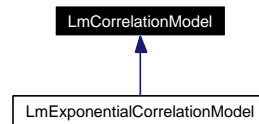
`registerAsObserver` is left as a backdoor in case the programmer cannot guarantee that the object pointed to will remain alive for the whole lifetime of the handle—namely, it should be set to `false` when the passed shared pointer was created with `owns = false` (the latter should only happen in a controlled 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.



## 7.360 LmCorrelationModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp>
```

Inheritance diagram for LmCorrelationModel:



### 7.360.1 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

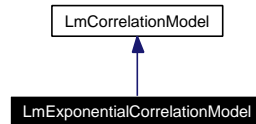
#### Protected Attributes

- const [Size](#) size\_
- std::vector< [Parameter](#) > arguments\_

## 7.361 LmExponentialCorrelationModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmexpcorrmodel.hpp>
```

Inheritance diagram for LmExponentialCorrelationModel:



### 7.361.1 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

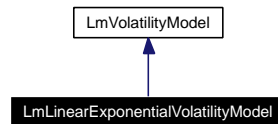
### Protected Member Functions

- `void generateArguments` ()

## 7.362 LmLinearExponentialVolatilityModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmlinepvolmodel.hpp>
```

Inheritance diagram for LmLinearExponentialVolatilityModel:



### 7.362.1 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))

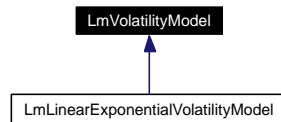
### Public Member Functions

- **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

## 7.363 LmVolatilityModel Class Reference

```
#include <ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp>
```

Inheritance diagram for LmVolatilityModel:



### 7.363.1 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

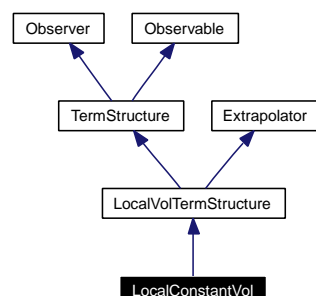
#### Protected Attributes

- const [Size](#) size\_
- std::vector< [Parameter](#) > arguments\_

## 7.364 LocalConstantVol Class Reference

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

Inheritance diagram for LocalConstantVol:



### 7.364.1 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 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*

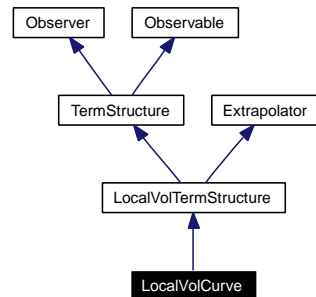
#### Visitability

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

## 7.365 LocalVolCurve Class Reference

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

Inheritance diagram for LocalVolCurve:



### 7.365.1 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 reference date, i.e., the date at which discount = 1*
- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Date](#) [maxDate](#) () const  
*the latest date for which the 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*

#### Visitability

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

#### Protected Member Functions

- [Volatility](#) [localVolImpl](#) ([Time](#), [Real](#)) const

## 7.365.2 Member Function Documentation

### 7.365.2.1 [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

$$\sigma_L(t) = \sqrt{\frac{d}{dt} \sigma_B^2(t) t}$$

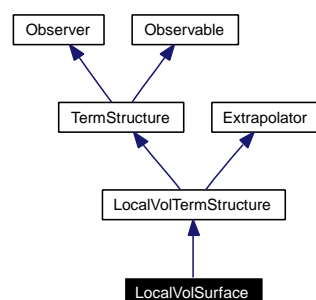
can be deduced which is here implemented.

Implements [LocalVolTermStructure](#).

## 7.366 LocalVolSurface Class Reference

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

Inheritance diagram for LocalVolSurface:



### 7.366.1 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 reference date, i.e., the date at which discount = 1*
- [DayCounter](#) [dayCounter](#) () const  
*the day counter used for date/time conversion*
- [Date](#) [maxDate](#) () const  
*the latest date for which the 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*

#### Visitability

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

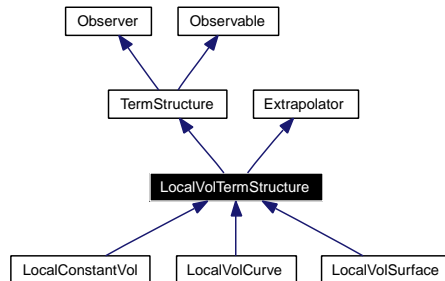
#### Protected Member Functions

- [Volatility localVolImpl](#) ([Time](#), [Real](#)) const  
*local vol calculation*

## 7.367 LocalVolTermStructure Class Reference

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

Inheritance diagram for LocalVolTermStructure:



### 7.367.1 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

### Constructors

See the *TermStructure* documentation for issues regarding constructors.

- [LocalVolTermStructure](#) ()  
*default constructor*
- [LocalVolTermStructure](#) (const [Date](#) &referenceDate)  
*initialize with a fixed reference date*
- [LocalVolTermStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)  
*calculate the reference date based on the global evaluation date*

### 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 [Date](#) [maxDate](#) () const =0  
*the latest date for which the term structure can return vols*
- [Time](#) [maxTime](#) () const  
*the latest time for which the term structure can return vols*

- 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*

## 7.367.2 Constructor & Destructor Documentation

### 7.367.2.1 [LocalVolTermStructure](#) ()

default constructor

#### [Warning](#)

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

## 7.368 LogLinear Class Reference

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

### 7.368.1 Detailed Description

log-linear interpolation factory and traits

#### Public Types

- enum { **global** = 0 }

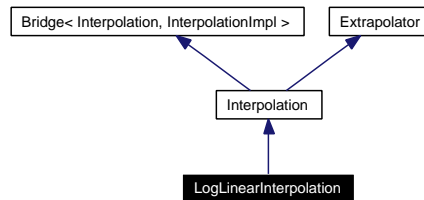
#### Public Member Functions

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

## 7.369 LogLinearInterpolation Class Reference

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

Inheritance diagram for LogLinearInterpolation:



### 7.369.1 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)`

### 7.369.2 Constructor & Destructor Documentation

#### 7.369.2.1 [LogLinearInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

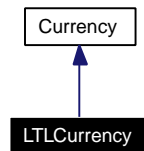
##### Precondition:

the *x* values must be sorted.

## 7.370 LTLCurrency Class Reference

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

Inheritance diagram for LTLCurrency:



### 7.370.1 Detailed Description

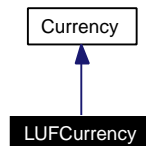
Lithuanian litas.

The ISO three-letter code is `LTL`; the numeric code is 440. It is divided in 100 centu.

## 7.371 LUFCurrency Class Reference

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

Inheritance diagram for LUFCurrency:



### 7.371.1 Detailed Description

Luxembourg franc.

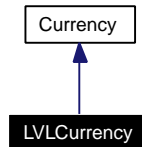
The ISO three-letter code was LUF; the numeric code was 442. It was divided in 100 centimes.

Obsoleted by the Euro since 1999.

## 7.372 LVLCurrency Class Reference

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

Inheritance diagram for LVLCurrency:



### 7.372.1 Detailed Description

Latvian lat.

The ISO three-letter code is LVL; the numeric code is 428. It is divided in 100 santims.



## 7.373 MakeMCDigitalEngine Class Template Reference

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

### 7.373.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MakeMCDigital-  
Engine< 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`

## 7.374 MakeMCEuropeanEngine Class Template Reference

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

### 7.374.1 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

## 7.375 MakeMCEuropeanHestonEngine Class Template Reference

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

### 7.375.1 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

## 7.376 MakeSchedule Class Reference

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

### 7.376.1 Detailed Description

helper class

This class provides a more comfortable interface to the argument list of Schedule's constructor.

### Public Member Functions

- **MakeSchedule** (const [Calendar](#) &calendar, const [Date](#) &startDate, const [Date](#) &endDate, [Frequency](#) frequency, [BusinessDayConvention](#) convention)
- **MakeSchedule** & **withStubDate** (const [Date](#) &d)
- **MakeSchedule** & **backwards** (bool flag=true)
- **MakeSchedule** & **forwards** (bool flag=true)
- **MakeSchedule** & **longFinalPeriod** (bool flag=true)
- **MakeSchedule** & **shortFinalPeriod** (bool flag=true)
- **operator Schedule** ()

## 7.377 Matrix Class Reference

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

### 7.377.1 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.

### 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

#### 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](#) > &)

#### 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
- `row_iterator` **operator[]** (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`)

*Returns the pseudo square root of a real symmetric matrix.*

- const `Disposable< Matrix > rankReducedSqrt` (const `Matrix` &, `Size` maxRank, `Real` componentRetainedPercentage, `SalvagingAlgorithm::Type`)

## 7.377.2 Member Function Documentation

### 7.377.2.1 const `Matrix` & operator+= (const `Matrix` &)

#### Precondition:

all matrices involved in an algebraic expression must have the same size.

## 7.377.3 Friends And Related Function Documentation

### 7.377.3.1 const `Disposable< Matrix > pseudoSqrt` (const `Matrix` &, `SalvagingAlgorithm::Type`) [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 Hypersphere decomposition:
  1. Jäckel "Monte Carlo Methods in Finance", Chapter 6
  2. Brigo "A Note on Correlation and Rank Reduction"
  3. Rapisarda, Brigo, Mercurio "Parameterizing correlations: a geometric interpretation"
- 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.

7.377.3.2 `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real componentRetainedPercentage, SalvagingAlgorithm::Type)` [related]

**Precondition:**

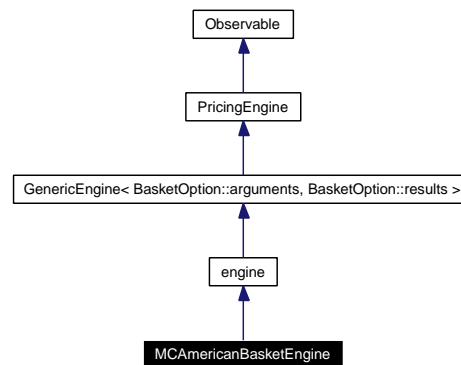
the given matrix must be symmetric.



## 7.378 MCAmericanBasketEngine Class Reference

```
#include <ql/PricingEngines/Basket/mcamericanbasketengine.hpp>
```

Inheritance diagram for MCAmericanBasketEngine:



### 7.378.1 Detailed Description

least-square Monte Carlo engine

#### Bug

This method is intrinsically weak for out-of-the-money options.

#### Bug

this engine does not yet work for put options. More problems might surface.

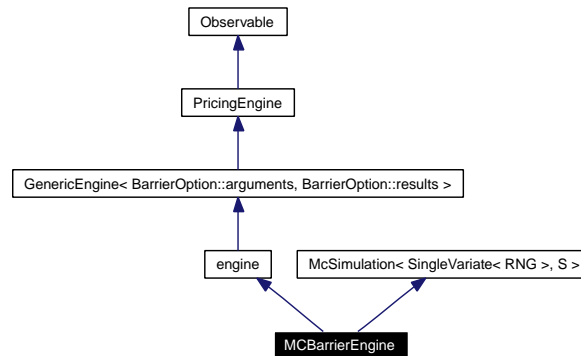
### Public Member Functions

- **MCAmericanBasketEngine** ([Size](#) requiredSamples, [Size](#) timeSteps, [BigNatural](#) seed=0, [bool](#) antitheticSampling=false)
- void **calculate** () const

## 7.379 MBarrierEngine Class Template Reference

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

Inheritance diagram for MBarrierEngine:



### 7.379.1 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`

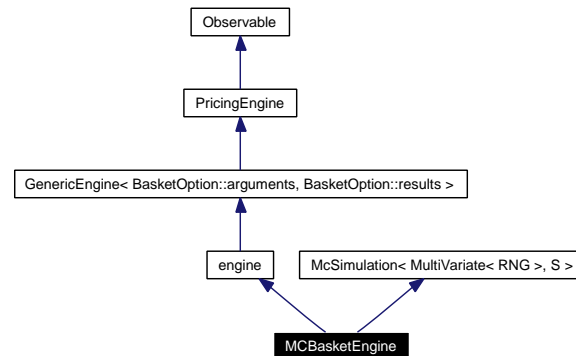
### Protected Attributes

- `Size maxTimeStepsPerYear_`
- `Size requiredSamples_`
- `Size maxSamples_`
- `Real requiredTolerance_`
- `bool isBiased_`
- `bool brownianBridge_`
- `BigNatural seed_`

## 7.380 MCBasketEngine Class Template Reference

```
#include <ql/PricingEngines/Basket/mcbasketengine.hpp>
```

Inheritance diagram for MCBasketEngine:



### 7.380.1 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

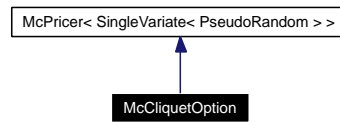
## Protected Attributes

- [Size](#) maxTimeStepsPerYear\_
- [Size](#) requiredSamples\_
- [Size](#) maxSamples\_
- [Real](#) requiredTolerance\_
- bool brownianBridge\_
- [BigNatural](#) seed\_

## 7.381 McCliquetOption Class Reference

```
#include <ql/Pricers/mccliquetoption.hpp>
```

Inheritance diagram for McCliquetOption:



### 7.381.1 Detailed Description

simple example of Monte Carlo pricer

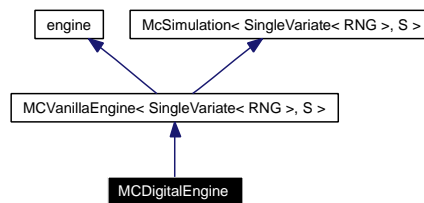
#### Public Member Functions

- **McCliquetOption** (Option::Type type, [Real](#) underlying, [Real](#) moneyness, 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)

## 7.382 MCDigitalEngine Class Template Reference

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

Inheritance diagram for MCDigitalEngine:



### 7.382.1 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)

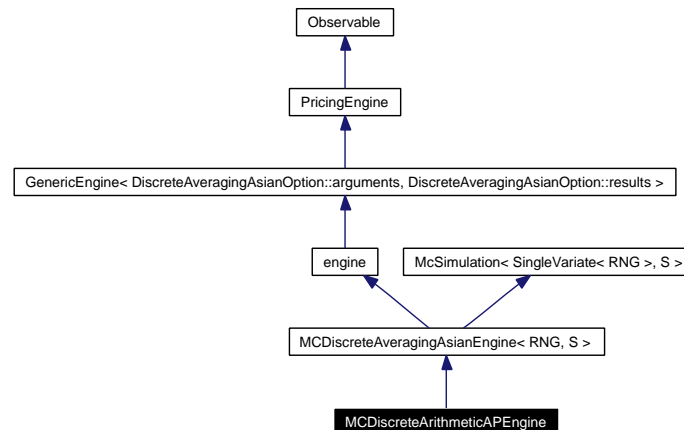
### Protected Member Functions

- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const

## 7.383 MCDiscreteArithmeticAPEngine Class Template Reference

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

Inheritance diagram for MCDiscreteArithmeticAPEngine:



### 7.383.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscrete-
ArithmeticAPEngine< 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](#))



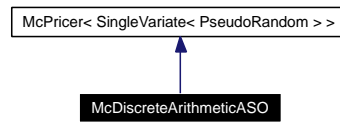
## Protected Member Functions

- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const
- boost::shared\_ptr< path\_pricer\_type > **controlPathPricer** () const
- boost::shared\_ptr< [PricingEngine](#) > **controlPricingEngine** () const

## 7.384 McDiscreteArithmeticASO Class Reference

```
#include <ql/Pricers/mcdiscretearithmeticaso.hpp>
```

Inheritance diagram for McDiscreteArithmeticASO:



### 7.384.1 Detailed Description

example of Monte Carlo pricer using a control variate.

#### Todo

continous-averaging version

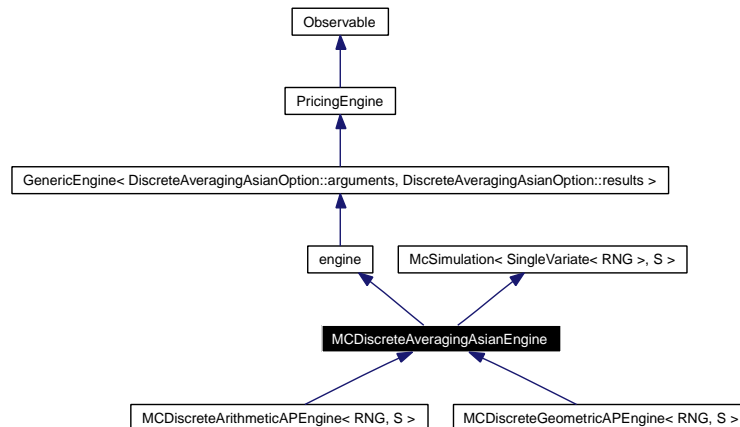
### Public Member Functions

- **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)

## 7.385 MCDiscreteAveragingAsianEngine Class Template Reference

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

Inheritance diagram for MCDiscreteAveragingAsianEngine:



### 7.385.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscreteAveragingAsianEngine< 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`

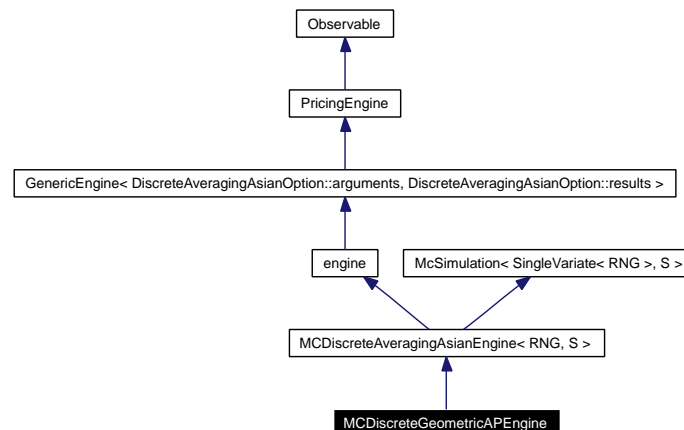
### Protected Attributes

- `Size maxTimeStepsPerYear_`
- `Size requiredSamples_`
- `Size maxSamples_`
- `Real requiredTolerance_`
- `bool brownianBridge_`
- `BigNatural seed_`

## 7.386 MCDiscreteGeometricAPEngine Class Template Reference

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

Inheritance diagram for MCDiscreteGeometricAPEngine:



### 7.386.1 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)

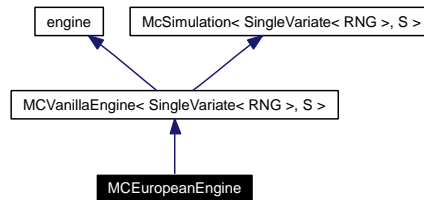
### Protected Member Functions

- boost::shared\_ptr< path\_pricer\_type > **pathPricer** () const

## 7.387 MCEuropeanEngine Class Template Reference

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

Inheritance diagram for MCEuropeanEngine:



### 7.387.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCEuropeanEngine< 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)

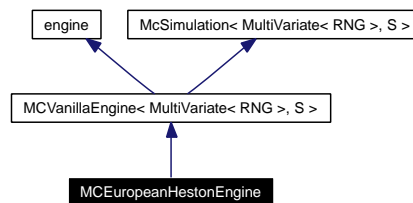
### Protected Member Functions

- boost::shared\_ptr< [path\\_pricer\\_type](#) > **pathPricer** () const

## 7.388 MCEuropeanHestonEngine Class Template Reference

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

Inheritance diagram for MCEuropeanHestonEngine:



### 7.388.1 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)

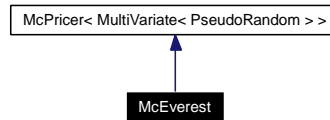
### Protected Member Functions

- `boost::shared_ptr< path_pricer_type > pathPricer () const`

## 7.389 McEverest Class Reference

```
#include <ql/Pricers/mceverest.hpp>
```

Inheritance diagram for McEverest:



### 7.389.1 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

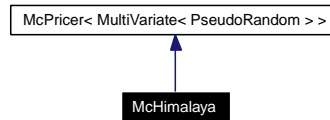
- **McEverest** (const std::vector< [Handle](#)< [YieldTermStructure](#) > > &dividendYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const std::vector< [Handle](#)< [BlackVolTermStructure](#) > > &volatilities, const [Matrix](#) &correlation, [Time](#) residualTime, [BigNatural](#) seed=0)



## 7.390 McHimalaya Class Reference

```
#include <ql/Pricers/mchimalaya.hpp>
```

Inheritance diagram for McHimalaya:



### 7.390.1 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.

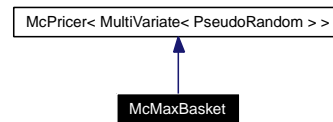
### Public Member Functions

- **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)

## 7.391 McMaxBasket Class Reference

```
#include <ql/Pricers/mcmaxbasket.hpp>
```

Inheritance diagram for McMaxBasket:



### 7.391.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

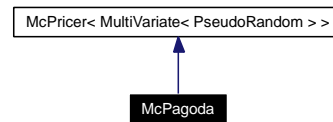
#### Public Member Functions

- **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)

## 7.392 McPagoda Class Reference

```
#include <ql/Pricers/mcpagoda.hpp>
```

Inheritance diagram for McPagoda:



### 7.392.1 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.

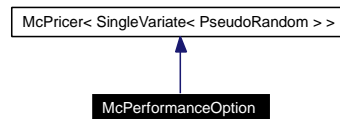
### Public Member Functions

- **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)

## 7.393 McPerformanceOption Class Reference

```
#include <ql/Pricers/mcperformanceoption.hpp>
```

Inheritance diagram for McPerformanceOption:



### 7.393.1 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](#) moneyness, const [Handle](#)< [YieldTermStructure](#) > &dividendYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > &times, [BigNatural](#) seed=0)

## 7.394 McPricer Class Template Reference

```
#include <ql/Pricers/mcpricer.hpp>
```

### 7.394.1 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](#) maxSample=QL\_MAX\_INTEGER) const  
*add samples until the required tolerance is reached*
- [Real valueWithSamples](#) ([Size](#) samples) const  
*simulate a fixed number of samples*
- [Real errorEstimate](#) () const  
*error Estimated of the samples simulated so far*
- const S & [sampleAccumulator](#) (void) const  
*access to the sample accumulator for more statistics*

### Protected Attributes

- boost::shared\_ptr< [MonteCarloModel](#)< MC, S > > [mcModel\\_](#)

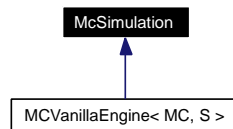
### Static Protected Attributes

- static const [Size](#) [minSample\\_](#) = 1023

## 7.395 McSimulation Class Template Reference

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

Inheritance diagram for McSimulation:



### 7.395.1 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](#) maxSample=QL\_MAX\_INTEGER) 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\_**

## Static Protected Attributes

- static const [Size](#) **minSample\_** = 1023

## 7.395.2 Member Function Documentation

7.395.2.1 void calculate ([Real](#) *requiredTolerance*, [Size](#) *requiredSamples*, [Size](#) *maxSamples*) const

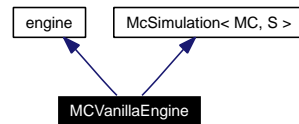
basic calculate method provided to inherited pricing engines

Initialize the one-factor Monte Carlo

## 7.396 MCVanillaEngine Class Template Reference

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

Inheritance diagram for MCVanillaEngine:



### 7.396.1 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\_



## 7.397 MersenneTwisterUniformRng Class Reference

```
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

### 7.397.1 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*

### 7.397.2 Constructor & Destructor Documentation

7.397.2.1 [MersenneTwisterUniformRng](#) (unsigned long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

### 7.397.3 Member Function Documentation

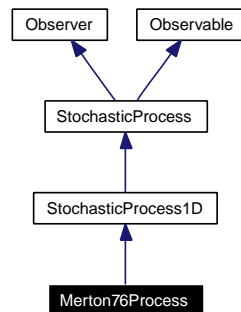
7.397.3.1 [sample\\_type next](#) () const

returns a sample with weight 1.0 containing a random number on (0.0, 1.0)-real-interval

## 7.398 Merton76Process Class Reference

```
#include <ql/Processes/merton76process.hpp>
```

Inheritance diagram for Merton76Process:



### 7.398.1 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](#) x0, [Real](#) dx) 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

## 7.398.2 Member Function Documentation

### 7.398.2.1 **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](#).

### 7.398.2.2 **Time** time (const **Date** &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

**Note:**

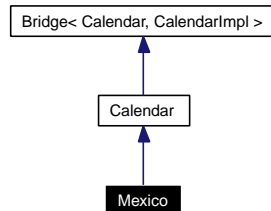
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

## 7.399 Mexico Class Reference

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

Inheritance diagram for Mexico:



### 7.399.1 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)

## 7.399.2 Member Enumeration Documentation

### 7.399.2.1 enum [Market](#)

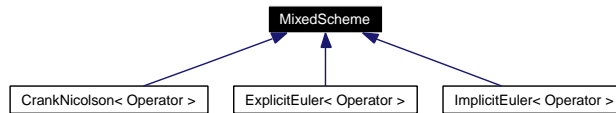
Enumerator:

*BMV* Mexican stock exchange.

## 7.400 MixedScheme Class Template Reference

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

Inheritance diagram for MixedScheme:



### 7.400.1 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\_**

## 7.401 Money Class Reference

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

### 7.401.1 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

#### Constructors

- `Money` (const `Currency` &currency, `Decimal` value)
- `Money` (`Decimal` value, const `Currency` &currency)

#### 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 &)`

## 7.401.2 Member Enumeration Documentation

### 7.401.2.1 enum `ConversionType`

Enumerator:

*NoConversion* do not perform conversions

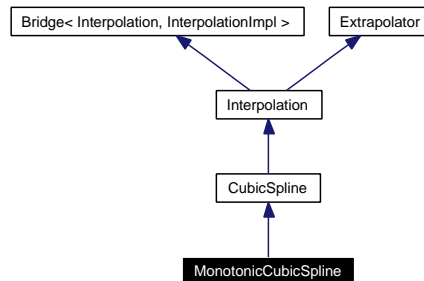
*BaseCurrencyConversion* convert both operands to the base currency before converting

*AutomatedConversion* return the result in the currency of the first operand

## 7.402 MonotonicCubicSpline Class Reference

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

Inheritance diagram for MonotonicCubicSpline:



### 7.402.1 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)`

### 7.402.2 Constructor & Destructor Documentation

- 7.402.2.1 `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.

## 7.403 MonteCarloModel Class Template Reference

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

### 7.403.1 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

## 7.404 MoreGreeks Class Reference

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

Inheritance diagram for MoreGreeks:



### 7.404.1 Detailed Description

more additional option results

#### Public Member Functions

- `void reset ()`

#### Public Attributes

- [Real](#) itmCashProbability
- [Real](#) deltaForward
- [Real](#) elasticity
- [Real](#) thetaPerDay
- [Real](#) strikeSensitivity

## 7.405 MoroInverseCumulativeNormal Class Reference

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

### 7.405.1 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](#)

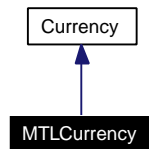
### Public Member Functions

- **MoroInverseCumulativeNormal** ([Real](#) average=0.0, [Real](#) sigma=1.0)
- **Real operator()** ([Real](#) x) const

## 7.406 MTLCurrency Class Reference

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

Inheritance diagram for MTLCurrency:



### 7.406.1 Detailed Description

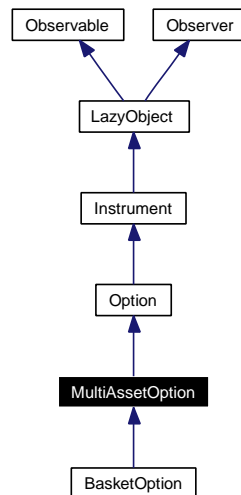
Maltese lira.

The ISO three-letter code is MTL; the numeric code is 470. It is divided in 100 cents.

## 7.407 MultiAssetOption Class Reference

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

Inheritance diagram for MultiAssetOption:



### 7.407.1 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*

## 7.407.2 Member Function Documentation

### 7.407.2.1 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](#).

### 7.407.2.2 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](#).

### 7.407.2.3 void [setupExpired](#) () const [protected, virtual]

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

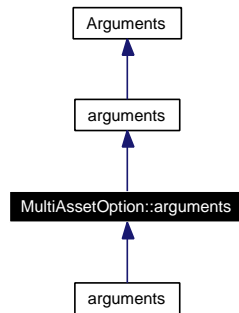
Reimplemented from [Instrument](#).



## 7.408 MultiAssetOption::arguments Class Reference

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

Inheritance diagram for MultiAssetOption::arguments:



### 7.408.1 Detailed Description

Arguments for multi-asset option calculation

#### Public Member Functions

- `void validate () const`

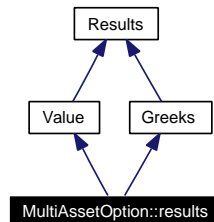
#### Public Attributes

- `boost::shared_ptr< StochasticProcess > stochasticProcess`

## 7.409 MultiAssetOption::results Class Reference

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

Inheritance diagram for MultiAssetOption::results:



### 7.409.1 Detailed Description

Results from multi-asset option calculation

#### Public Member Functions

- `void reset ()`

## 7.410 MultiCubicSpline Class Template Reference

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

### 7.410.1 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

## 7.411 MultiPath Class Reference

```
#include <ql/MonteCarlo/multipath.hpp>
```

### 7.411.1 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
- [Path](#) & `operator[] (Size j)`

## 7.412 MultiPathGenerator Class Template Reference

```
#include <ql/MonteCarlo/multipathgenerator.hpp>
```

### 7.412.1 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

## 7.413 MultiVariate Struct Template Reference

```
#include <ql/MonteCarlo/mctraits.hpp>
```

### 7.413.1 Detailed Description

**template<class RNG = PseudoRandom> struct QuantLib::MultiVariate< RNG >**

default Monte Carlo traits for multi-variate models

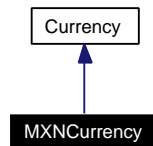
#### Public Types

- 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**
- enum { **allowsErrorEstimate** = RNG::allowsErrorEstimate }

## 7.414 MXNCurrency Class Reference

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

Inheritance diagram for MXNCurrency:



### 7.414.1 Detailed Description

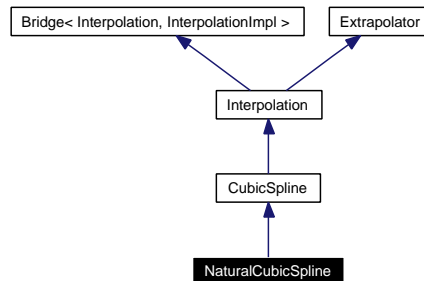
Mexican peso.

The ISO three-letter code is MXN; the numeric code is 484. It is divided in 100 centavos.

## 7.415 NaturalCubicSpline Class Reference

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

Inheritance diagram for NaturalCubicSpline:



### 7.415.1 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)`

### 7.415.2 Constructor & Destructor Documentation

#### 7.415.2.1 [NaturalCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

**Precondition:**

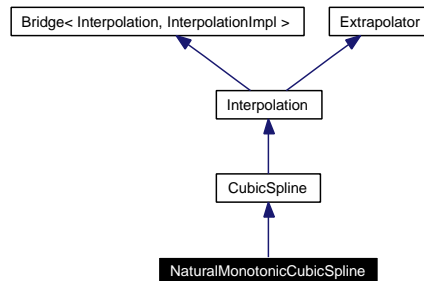
the *x* values must be sorted.



## 7.416 NaturalMonotonicCubicSpline Class Reference

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

Inheritance diagram for NaturalMonotonicCubicSpline:



### 7.416.1 Detailed Description

Natural cubic spline with monotonicity constraint.

#### Public Member Functions

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

### 7.416.2 Constructor & Destructor Documentation

7.416.2.1 [NaturalMonotonicCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

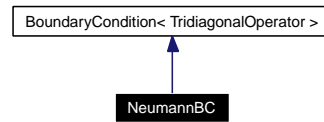
**Precondition:**

the *x* values must be sorted.

## 7.417 NeumannBC Class Reference

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

Inheritance diagram for NeumannBC:



### 7.417.1 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](#))

### 7.417.2 Member Function Documentation

#### 7.417.2.1 void applyBeforeApplying ([TridiagonalOperator](#) &) const [virtual]

This method modifies an operator  $L$  before it is applied to an array  $u$  so that  $v = Lu$  will satisfy the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

#### 7.417.2.2 void applyAfterApplying ([Array](#) &) const [virtual]

This method modifies an array  $u$  so that it satisfies the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

**7.417.2.3 void applyBeforeSolving ([TridiagonalOperator](#) &, [Array](#) & *rhs*) const** [virtual]

This method modifies an operator  $L$  before the linear system  $Lu' = u$  is solved so that  $u'$  will satisfy the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

**7.417.2.4 void applyAfterSolving ([Array](#) &) const** [virtual]

This method modifies an array  $u$  so that it satisfies the given condition.

Implements [BoundaryCondition< TridiagonalOperator >](#).

**7.417.2.5 void setTime ([Time](#))** [virtual]

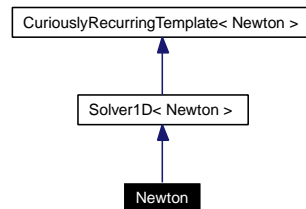
This method sets the current time for time-dependent boundary conditions.

Implements [BoundaryCondition< TridiagonalOperator >](#).

## 7.418 Newton Class Reference

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

Inheritance diagram for Newton:



### 7.418.1 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.

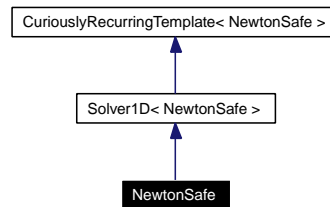
### Public Member Functions

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

## 7.419 NewtonSafe Class Reference

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

Inheritance diagram for NewtonSafe:



### 7.419.1 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.

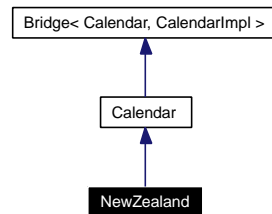
### Public Member Functions

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

## 7.420 NewZealand Class Reference

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

Inheritance diagram for NewZealand:



### 7.420.1 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)

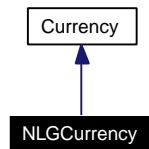
**Note:**

The holiday rules for New Zealand were documented by David Gilbert for IDB (<http://www.jrefinery.com/ibd/>)

## 7.421 NLGCurrency Class Reference

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

Inheritance diagram for NLGCurrency:



### 7.421.1 Detailed Description

Dutch guilder.

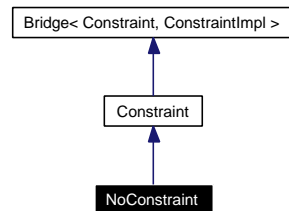
The ISO three-letter code was NLG; the numeric code was 528. It was divided in 100 cents.

Obsoleted by the Euro since 1999.

## 7.422 NoConstraint Class Reference

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

Inheritance diagram for NoConstraint:



### 7.422.1 Detailed Description

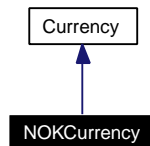
No constraint.



## 7.423 NOKCurrency Class Reference

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

Inheritance diagram for NOKCurrency:



### 7.423.1 Detailed Description

Norwegian krone.

The ISO three-letter code is NOK; the numeric code is 578. It is divided in 100 øre.

## 7.424 NonLinearLeastSquare Class Reference

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

### 7.424.1 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*

## 7.425 NormalDistribution Class Reference

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

### 7.425.1 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.

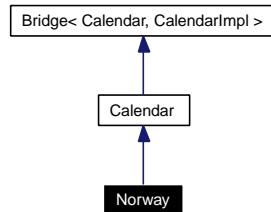
### Public Member Functions

- **NormalDistribution** ([Real](#) average=0.0, [Real](#) sigma=1.0)
- **Real operator()** ([Real](#)  $x$ ) const
- **Real derivative** ([Real](#)  $x$ ) const

## 7.426 Norway Class Reference

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

Inheritance diagram for Norway:



### 7.426.1 Detailed Description

Norwegian calendar.

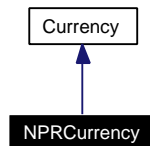
Holidays:

- 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

## 7.427 NPRCurrency Class Reference

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

Inheritance diagram for NPRCurrency:



### 7.427.1 Detailed Description

Nepal rupee.

The ISO three-letter code is NPR; the numeric code is 524. It is divided in 100 paise.

## 7.428 Null Class Template Reference

```
#include <ql/Utilities/null.hpp>
```

### 7.428.1 Detailed Description

```
template<class Type> class QuantLib::Null< Type >
```

template class providing a null value for a given type.

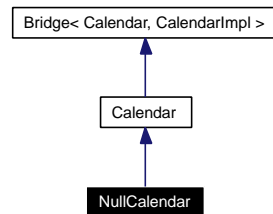
### Public Member Functions

- `operator Type () const`

## 7.429 NullCalendar Class Reference

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

Inheritance diagram for NullCalendar:



### 7.429.1 Detailed Description

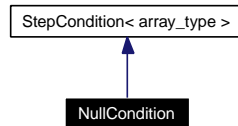
Calendar for reproducing theoretical calculations.

This calendar has no holidays. It ensures that dates at whole-month distances have the same day of month.

## 7.430 NullCondition Class Template Reference

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

Inheritance diagram for NullCondition:



### 7.430.1 Detailed Description

```
template<class array_type> class QuantLib::NullCondition< array_type >
```

null step condition

#### Public Member Functions

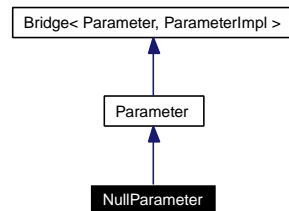
- void **applyTo** (array\_type &, [Time](#)) const



## 7.431 NullParameter Class Reference

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

Inheritance diagram for NullParameter:



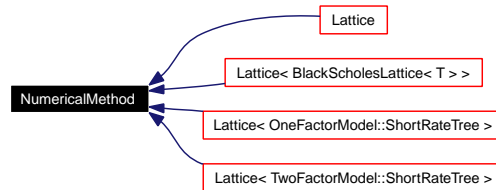
### 7.431.1 Detailed Description

Parameter which is always zero  $a(t) = 0$

## 7.432 NumericalMethod Class Reference

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

Inheritance diagram for NumericalMethod:



### 7.432.1 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\_

### 7.432.2 Member Function Documentation

#### 7.432.2.1 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< OneFactorModel::ShortRateTree >](#), [Lattice< TwoFactorModel::ShortRateTree >](#), and [Lattice< BlackScholesLattice< T > >](#).

**7.432.2.2 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:

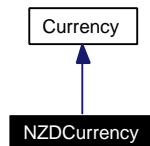
```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Implemented in [Lattice](#), [TsiveriotisFernandesLattice](#), [Lattice< OneFactorModel::ShortRateTree >](#), [Lattice< TwoFactorModel::ShortRateTree >](#), and [Lattice< BlackScholesLattice< T > >](#).

## 7.433 NZDCurrency Class Reference

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

Inheritance diagram for NZDCurrency:



### 7.433.1 Detailed Description

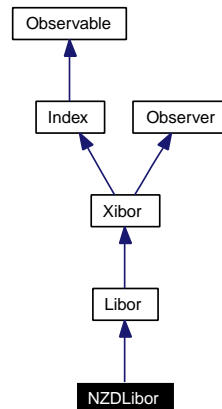
New Zealand dollar.

The ISO three-letter code is NZD; the numeric code is 554. It is divided in 100 cents.

## 7.434 NZDLibor Class Reference

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

Inheritance diagram for NZDLibor:



### 7.434.1 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>>.

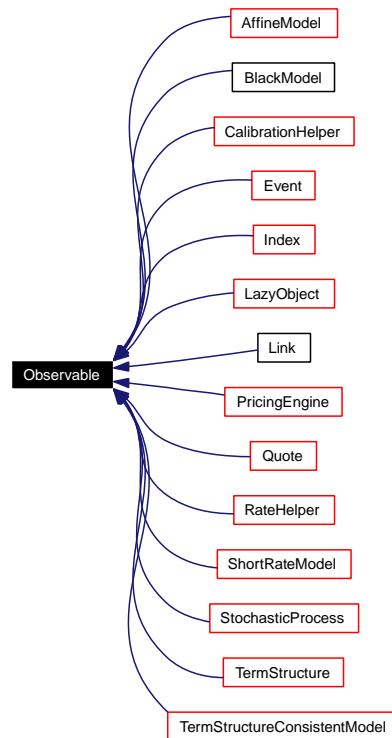
### Public Member Functions

- **NZDLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.435 Observable Class Reference

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

Inheritance diagram for Observable:



### 7.435.1 Detailed Description

Object that notifies its changes to a set of observables.

#### Public Member Functions

- void [notifyObservers\(\)](#)

#### Friends

- class [Observer](#)

### 7.435.2 Member Function Documentation

#### 7.435.2.1 void notifyObservers()

This method should be called at the end of non-const methods or when the programmer desires to notify any changes.

## 7.436 ObservableValue Class Template Reference

```
#include <ql/Utilities/observablevalue.hpp>
```

### 7.436.1 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 &)
- **operator T** () const  
*implicit conversion*
- const T & **value** () const  
*explicit inspector*

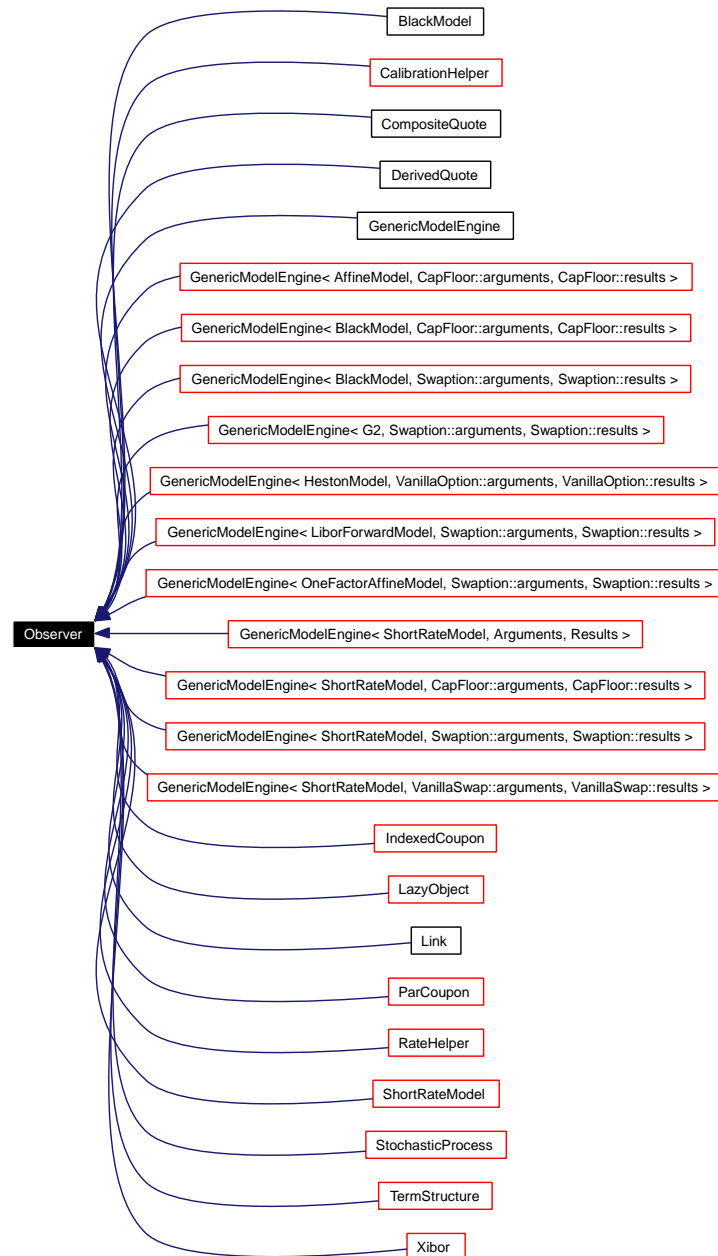
### controlled assignment

- **ObservableValue**< T > & **operator=** (const T &)
- **ObservableValue**< T > & **operator=** (const **ObservableValue**< T > &)

## 7.437 Observer Class Reference

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

Inheritance diagram for Observer:



### 7.437.1 Detailed Description

Object that gets notified when a given observable changes.



## Public Member Functions

- **Observer** (const [Observer](#) &)
- **Observer** & **operator=** (const [Observer](#) &)
- template<class T> void **registerWith** (const boost::shared\_ptr< T > &h)
- template<class T> void **unregisterWith** (const boost::shared\_ptr< T > &h)
- virtual void **update** ()=0

## 7.437.2 Member Function Documentation

### 7.437.2.1 virtual void update () [pure virtual]

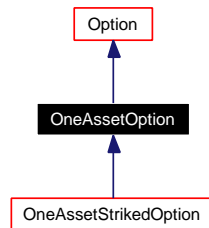
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implemented in [IndexedCoupon](#), [ParCoupon](#), [Link](#), [Xibor](#), [LazyObject](#), [BlackModel](#), [GenericModelEngine](#), [LatticeShortRateModelEngine](#), [BlackScholesProcess](#), [DerivedQuote](#), [CompositeQuote](#), [CalibrationHelper](#), [ShortRateModel](#), [StochasticProcess](#), [TermStructure](#), [AffineTermStructure](#), [ExtendedDiscountCurve](#), [FlatForward](#), [PiecewiseYieldCurve](#), [RateHelper](#), [CapVolatilityVector](#), [GenericModelEngine< ShortRateModel, Arguments, Results >](#), [GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results >](#), [GenericModelEngine< LiborForwardModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< ShortRateModel, VanillaSwap::arguments, VanillaSwap::results >](#), [GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< G2, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >](#), [GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >](#), [GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< HestonModel, VanillaOption::arguments, VanillaOption::results >](#), [LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >](#), [LatticeShortRateModelEngine< VanillaSwap::arguments, VanillaSwap::results >](#), and [LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >](#).

## 7.438 OneAssetOption Class Reference

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

Inheritance diagram for OneAssetOption:



### 7.438.1 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*

## 7.438.2 Member Function Documentation

- 7.438.2.1 [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.

- 7.438.2.2 `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](#), [OneAssetStrikedOption](#), [QuantoForwardVanillaOption](#), and [QuantoVanillaOption](#).

**7.438.2.3 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](#).

**7.438.2.4 void setupExpired () const** [protected, virtual]

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

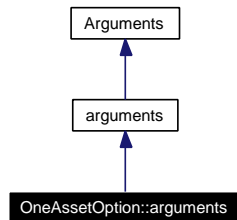
Reimplemented from [Instrument](#).

Reimplemented in [OneAssetStrikedOption](#), and [QuantoVanillaOption](#).

## 7.439 OneAssetOption::arguments Class Reference

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

Inheritance diagram for OneAssetOption::arguments:



### 7.439.1 Detailed Description

Arguments for single-asset option calculation

#### Public Member Functions

- void **validate** () const

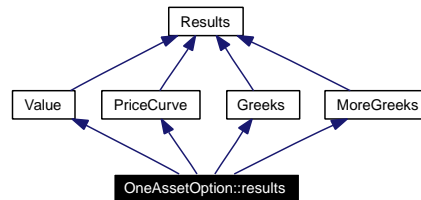
#### Public Attributes

- boost::shared\_ptr< [StochasticProcess](#) > **stochasticProcess**

## 7.440 OneAssetOption::results Class Reference

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

Inheritance diagram for OneAssetOption::results:



### 7.440.1 Detailed Description

Results from single-asset option calculation

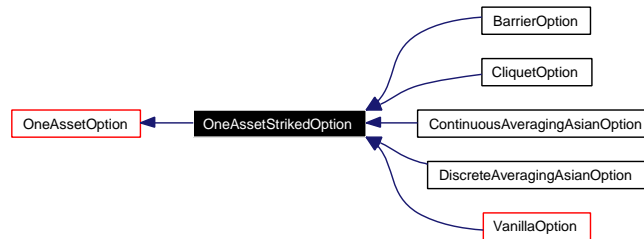
#### Public Member Functions

- void `reset()`

## 7.441 OneAssetStrikedOption Class Reference

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

Inheritance diagram for OneAssetStrikedOption:



### 7.441.1 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\\_](#)

### 7.441.2 Member Function Documentation

#### 7.441.2.1 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](#), [QuantoForwardVanillaOption](#), and [QuantoVanillaOption](#).

**7.441.2.2 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](#).

**7.441.2.3 void setupExpired () const** [protected, virtual]

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

Reimplemented from [OneAssetOption](#).

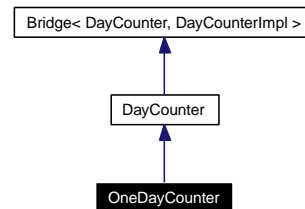
Reimplemented in [QuantoVanillaOption](#).



## 7.442 OneDayCounter Class Reference

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

Inheritance diagram for OneDayCounter:



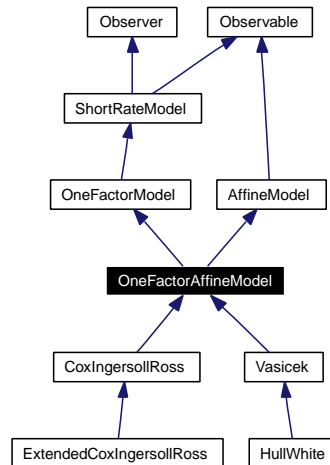
### 7.442.1 Detailed Description

1/1 day count convention

## 7.443 OneFactorAffineModel Class Reference

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

Inheritance diagram for OneFactorAffineModel:



### 7.443.1 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.*

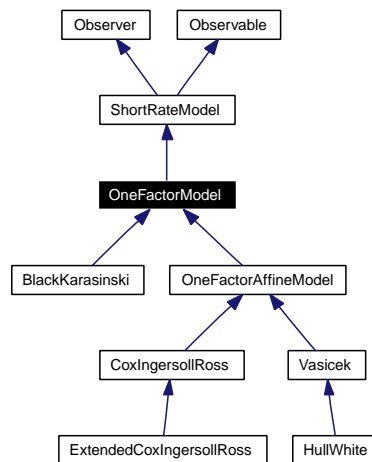
### Protected Member Functions

- virtual **Real** **A** ([Time](#) t, [Time](#) T) const =0
- virtual **Real** **B** ([Time](#) t, [Time](#) T) const =0

## 7.444 OneFactorModel Class Reference

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

Inheritance diagram for OneFactorModel:



### 7.444.1 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.*

## 7.445 OneFactorModel::ShortRateDynamics Class Reference

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

### 7.445.1 Detailed Description

Base class describing the short-rate dynamics.

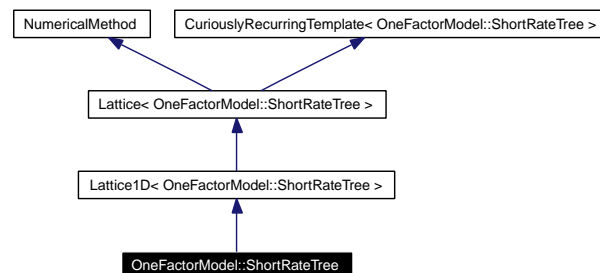
#### Public Member Functions

- **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.*

## 7.446 OneFactorModel::ShortRateTree Class Reference

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

Inheritance diagram for OneFactorModel::ShortRateTree:



### 7.446.1 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

## 7.447 OperatorFactory Class Reference

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

### 7.447.1 Detailed Description

Black-Scholes-Merton differential operator.

#### Tests

coefficients are tested against constant BSM operator

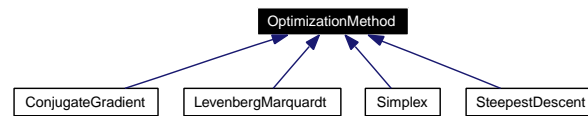
### Static Public Member Functions

- static [TridiagonalOperator](#) **getOperator** (const boost::shared\_ptr< [BlackScholesProcess](#) > &process, const [Array](#) &grid, [Time](#) residualTime, bool timeDependent)
- static [TridiagonalOperator](#) **getOperator** (const boost::shared\_ptr< [OneFactorModel::ShortRateDynamics](#) > &process, const [Array](#) &grid)

## 7.448 OptimizationMethod Class Reference

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

Inheritance diagram for OptimizationMethod:



### 7.448.1 Detailed Description

Abstract class for constrained optimization method.

#### Public Member Functions

- 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

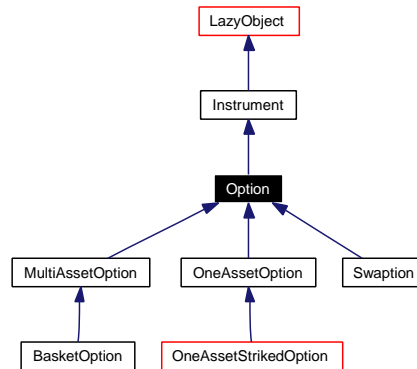
- [Array initialValue\\_](#)  
*initial value of unknowns*
- [Integer iterationNumber\\_](#)  
*current iteration step in the Optimization process*
- [EndCriteria endCriteria\\_](#)  
*optimization end criteria*
- [Integer functionEvaluation\\_](#)  
*number of evaluation of cost function and its gradient*
- [Integer gradientEvaluation\\_](#)
- [Real functionValue\\_](#)  
*function and gradient norm values of the last step*
- [Real squaredNorm\\_](#)
- [Array x\\_](#)  
*current values of the local minimum and the search direction*
- [Array searchDirection\\_](#)



## 7.449 Option Class Reference

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

Inheritance diagram for Option:



### 7.449.1 Detailed Description

base option class

#### Public Types

- enum Type { Call, Put }

#### 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)

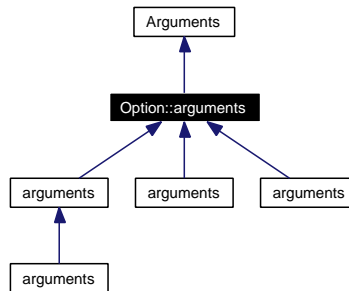
#### Classes

- class [arguments](#)

## 7.450 Option::arguments Class Reference

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

Inheritance diagram for Option::arguments:



### 7.450.1 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

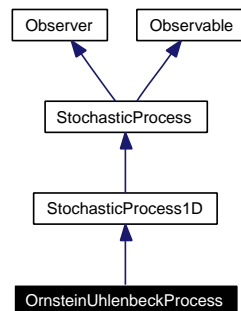
### Public Attributes

- boost::shared\_ptr< [Payoff](#) > **payoff**
- boost::shared\_ptr< [Exercise](#) > **exercise**
- std::vector< [Time](#) > **stoppingTimes**

## 7.451 OrnsteinUhlenbeckProcess Class Reference

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

Inheritance diagram for OrnsteinUhlenbeckProcess:



### 7.451.1 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.  $\mu(t, x_t)$*
- [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

### 7.451.2 Member Function Documentation

#### 7.451.2.1 [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](#).

**7.451.2.2   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](#).

**7.451.2.3   Real variance (Time t0, Real x0, Time dt) const   [virtual]**

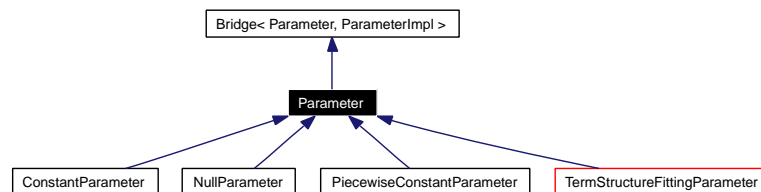
returns the variance  $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$  of the process after a time interval  $\Delta t$  according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

## 7.452 Parameter Class Reference

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

Inheritance diagram for Parameter:



### 7.452.1 Detailed Description

Base class for model arguments.

#### Public Member Functions

- const [Array](#) & **params** () const
- void **setParam** ([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\_**

## 7.453 ParameterImpl Class Reference

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

### 7.453.1 Detailed Description

Base class for model parameter implementation.

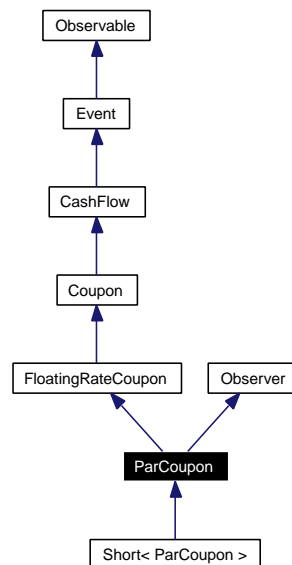
#### Public Member Functions

- virtual [Real](#) value (const [Array](#) &params, [Time](#) t) const =0

## 7.454 ParCoupon Class Reference

```
#include <ql/CashFlows/parcoupon.hpp>
```

Inheritance diagram for ParCoupon:



### 7.454.1 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** (*Real* nominal, const *Date* &paymentDate, const boost::shared\_ptr< *Xibor* > &index, const *Date* &startDate, const *Date* &endDate, *Integer* fixingDays, *Spread* spread=0.0, const *Date* &refPeriodStart=*Date*(), const *Date* &refPeriodEnd=*Date*(), const *DayCounter* &dayCounter=*DayCounter*())

#### CashFlow interface

- *Real* amount () const  
*returns the amount of the cash flow*

#### Coupon interface

- *DayCounter* dayCounter () const  
*day counter for accrual calculation*

### FloatingRateCoupon interface

- [Rate indexFixing](#) () const  
*fixing of the underlying index*
- [Date fixingDate](#) () const  
*fixing date*

### Inspectors

- const boost::shared\_ptr< [Xibor](#) > & [index](#) () const

### Observer interface

- void [update](#) ()

### Visitability

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

## 7.454.2 Member Function Documentation

### 7.454.2.1 [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](#).

Reimplemented in [Short< ParCoupon >](#).

### 7.454.2.2 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](#).



## 7.455 Path Class Reference

```
#include <ql/MonteCarlo/path.hpp>
```

### 7.455.1 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](#) & **operator[]** ([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*

## 7.456 PathGenerator Class Template Reference

```
#include <ql/MonteCarlo/pathgenerator.hpp>
```

### 7.456.1 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](#) timeSteps, const GSG &generator, bool brownianBridge)
- **PathGenerator** (const boost::shared\_ptr< [StochasticProcess](#) > &, const [TimeGrid](#) &timeGrid, const GSG &generator, bool brownianBridge)

#### inspectors

- const [sample\\_type](#) & **next** () const
- const [sample\\_type](#) & **antithetic** () const
- [Size](#) **size** () const
- const [TimeGrid](#) & **timeGrid** () const

## 7.457 PathPricer Class Template Reference

```
#include <ql/MonteCarlo/pathpricer.hpp>
```

### 7.457.1 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](#).

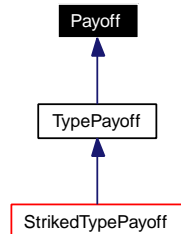
### Public Member Functions

- virtual ValueType **operator()** (const PathType &path) const =0

## 7.458 Payoff Class Reference

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

Inheritance diagram for Payoff:



### 7.458.1 Detailed Description

Base class for option payoffs.

#### Public Member Functions

##### Payoff interface

- virtual [Real](#) **operator()** ([Real](#) price) const =0

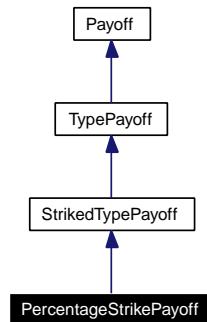
##### Visitability

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

## 7.459 PercentageStrikePayoff Class Reference

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

Inheritance diagram for PercentageStrikePayoff:



### 7.459.1 Detailed Description

Payoff with strike expressed as percentage

#### Public Member Functions

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

## 7.460 Period Class Reference

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

### 7.460.1 Detailed Description

Time period described by a number of a given time unit.

Examples:

[BermudanSwaption.cpp](#).

### Public Member Functions

- **Period** ([Integer](#) n, [TimeUnit](#) units)
- [Integer](#) **length** () const
- [TimeUnit](#) **units** () const

### Related Functions

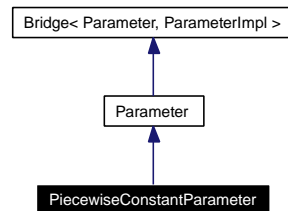
(Note that these are not member functions.)

- [Period](#) **operator** \* ([Integer](#) n, [TimeUnit](#) units)
- [Period](#) **operator** \* ([TimeUnit](#) units, [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](#) &)

## 7.461 PiecewiseConstantParameter Class Reference

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

Inheritance diagram for PiecewiseConstantParameter:



### 7.461.1 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

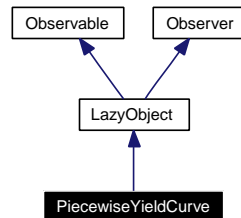
### Public Member Functions

- `PiecewiseConstantParameter` (const std::vector< [Time](#) > &times)

## 7.462 PiecewiseYieldCurve Class Template Reference

```
#include <ql/TermStructures/piecewiseyieldcurve.hpp>
```

Inheritance diagram for PiecewiseYieldCurve:



### 7.462.1 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

### Constructors

- **PiecewiseYieldCurve** (const [Date](#) &referenceDate, const std::vector< boost::shared\_ptr< [RateHelper](#) > > &instruments, const [DayCounter](#) &dayCounter, [Real](#) accuracy=1.0e-12, const Interpolator &i=Interpolator())
- **PiecewiseYieldCurve** ([Integer](#) settlementDays, const [Calendar](#) &calendar, const std::vector< boost::shared\_ptr< [RateHelper](#) > > &instruments, const [DayCounter](#) &dayCounter, [Real](#) accuracy=1.0e-12, const Interpolator &i=Interpolator())

### 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**

### 7.462.2 Member Function Documentation

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

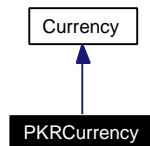
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [LazyObject](#).

## 7.463 PKRCurrency Class Reference

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

Inheritance diagram for PKRCurrency:



### 7.463.1 Detailed Description

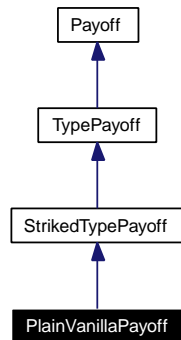
Pakistani rupee.

The ISO three-letter code is PKR; the numeric code is 586. It is divided in 100 paisa.

## 7.464 PlainVanillaPayoff Class Reference

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

Inheritance diagram for PlainVanillaPayoff:



### 7.464.1 Detailed Description

Plain-vanilla payoff.

**Examples:**

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

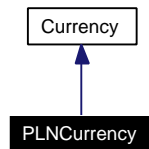
### Public Member Functions

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

## 7.465 PLNCurrency Class Reference

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

Inheritance diagram for PLNCurrency:



### 7.465.1 Detailed Description

Polish zloty.

The ISO three-letter code is PLN; the numeric code is 985. It is divided in 100 groszy.

## 7.466 PoissonDistribution Class Reference

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

### 7.466.1 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.

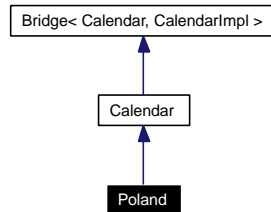
### Public Member Functions

- **PoissonDistribution** ([Real](#) mu)
- **[Real](#) operator()** (BigNatural k) const

## 7.467 Poland Class Reference

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

Inheritance diagram for Poland:



### 7.467.1 Detailed Description

Polish calendar.

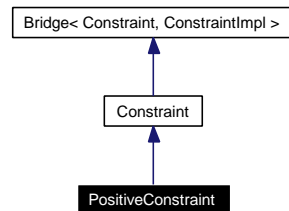
Holidays:

- 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

## 7.468 PositiveConstraint Class Reference

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

Inheritance diagram for PositiveConstraint:



### 7.468.1 Detailed Description

Constraint imposing positivity to all arguments

## 7.469 PriceCurve Class Reference

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

Inheritance diagram for PriceCurve:



### 7.469.1 Detailed Description

additional pricing results

#### Public Member Functions

- `void reset ()`

#### Public Attributes

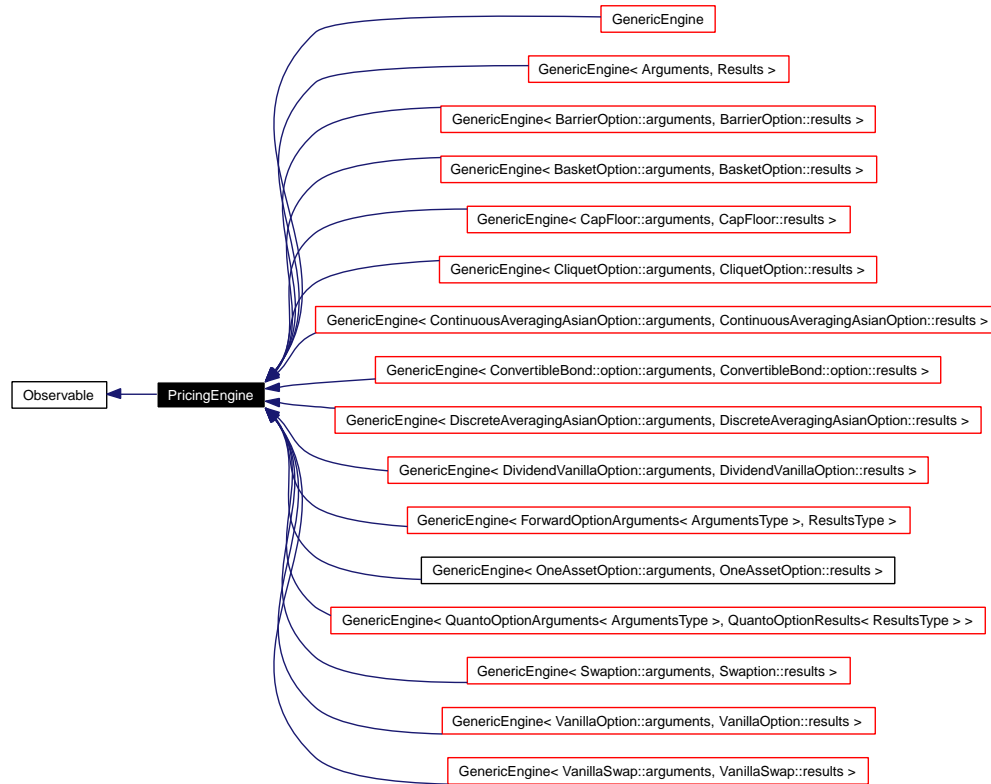
- [SampledCurve](#) `priceCurve`



## 7.470 PricingEngine Class Reference

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

Inheritance diagram for PricingEngine:



### 7.470.1 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

## 7.471 PrimeNumbers Class Reference

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

### 7.471.1 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.*

## 7.472 Problem Class Reference

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

### 7.472.1 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

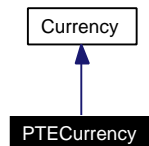
- [CostFunction](#) & [costFunction\\_](#)  
*Unconstrained cost function.*
- [Constraint](#) & [constraint\\_](#)  
*Constraint.*
- [OptimizationMethod](#) & [method\\_](#)

*constrained optimization method*

## 7.473 PTECurrency Class Reference

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

Inheritance diagram for PTECurrency:



### 7.473.1 Detailed Description

Portuguese escudo.

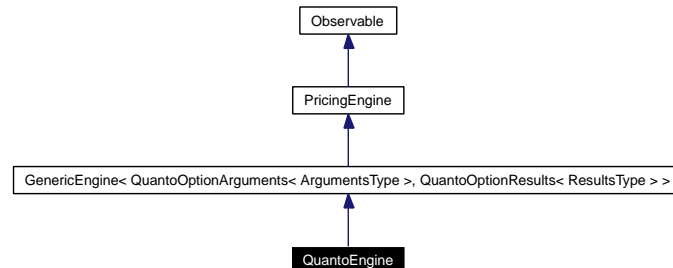
The ISO three-letter code was PTE; the numeric code was 620. It was divided in 100 centavos.

Obsoleted by the Euro since 1999.

## 7.474 QuantoEngine Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

Inheritance diagram for QuantoEngine:



### 7.474.1 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\_**

## 7.474.2 Member Function Documentation

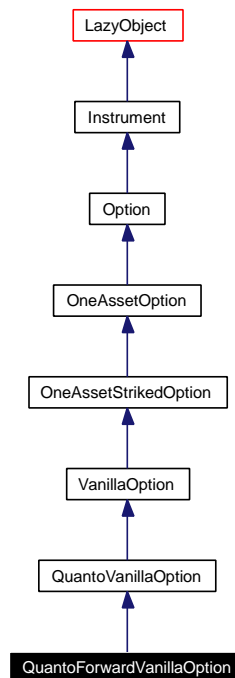
### 7.474.2.1 ArgumentsType\* underlyingArgs () const

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.

## 7.475 QuantoForwardVanillaOption Class Reference

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

Inheritance diagram for QuantoForwardVanillaOption:



### 7.475.1 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



## 7.475.2 Member Function Documentation

### 7.475.2.1 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](#).

## 7.476 QuantoOptionArguments Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

### 7.476.1 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`

## 7.477 QuantoOptionResults Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

### 7.477.1 Detailed Description

```
template<class ResultsType> class QuantLib::QuantoOptionResults< ResultsType >
```

Results from quanto option calculation

### Public Member Functions

- `void reset ()`

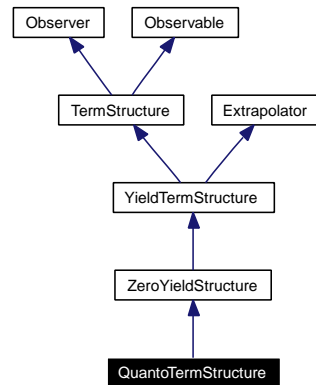
### Public Attributes

- [Real](#) `qvega`
- [Real](#) `qrho`
- [Real](#) `qlambda`

## 7.478 QuantoTermStructure Class Reference

```
#include <ql/TermStructures/quantotermstructure.hpp>
```

Inheritance diagram for QuantoTermStructure:



### 7.478.1 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 reference date, i.e., the date at which discount = 1*
- [Date](#) [maxDate](#) () const  
*the latest date for which the curve can return rates*

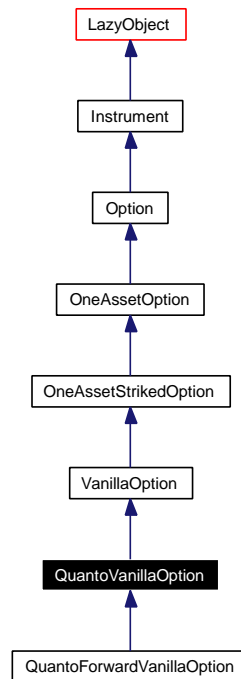
## Protected Member Functions

- [Rate zeroYieldImpl](#) ([Time](#)) const  
*returns the zero yield as seen from the evaluation date*

## 7.479 QuantoVanillaOption Class Reference

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

Inheritance diagram for QuantoVanillaOption:



### 7.479.1 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_`

## 7.479.2 Member Function Documentation

### 7.479.2.1 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](#).

### 7.479.2.2 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](#).

### 7.479.2.3 void `setupExpired` () const [protected, virtual]

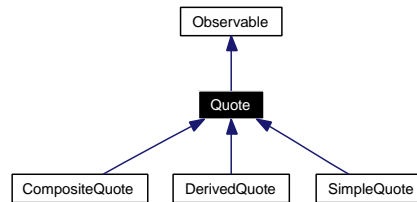
This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [OneAssetStrikedOption](#).

## 7.480 Quote Class Reference

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

Inheritance diagram for Quote:



### 7.480.1 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*



## 7.481 RandomizedLDS Class Template Reference

```
#include <ql/RandomNumbers/randomizedlds.hpp>
```

### 7.481.1 Detailed Description

```
template<class LDS, class PRS = RandomSequenceGenerator<MersenneTwisterUniformRng>> 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

## 7.481.2 Member Function Documentation

### 7.481.2.1 void nextRandomizer ()

update the randomizing vector and re-initialize the low discrepancy generator

## 7.482 RandomSequenceGenerator Class Template Reference

```
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
```

### 7.482.1 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**

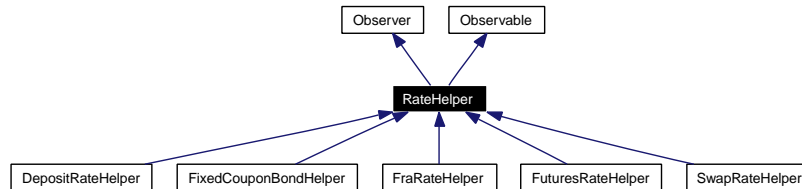
### Public Member Functions

- **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

## 7.483 RateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for RateHelper:



### 7.483.1 Detailed Description

Base class for rate helpers.

This class provides an abstraction for the instruments used to bootstrap a term structure. It is advised that a rate helper for an instrument contains an instance of the actual instrument class to ensure 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](#) contains a [Swap](#) 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](#) [latestDate](#) () const =0  
*latest relevant date*

#### Observer interface

- void [update](#) ()

### Protected Attributes

- [Handle](#)< [Quote](#) > [quote\\_](#)
- [YieldTermStructure](#) \* [termStructure\\_](#)

## 7.483.2 Member Function Documentation

### 7.483.2.1 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](#), [DepositRateHelper](#), [FraRateHelper](#), and [SwapRateHelper](#).

### 7.483.2.2 virtual [Date](#) latestDate () const [pure 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.

Implemented in [FixedCouponBondHelper](#), [DepositRateHelper](#), [FraRateHelper](#), [FuturesRateHelper](#), and [SwapRateHelper](#).

### 7.483.2.3 void update () [virtual]

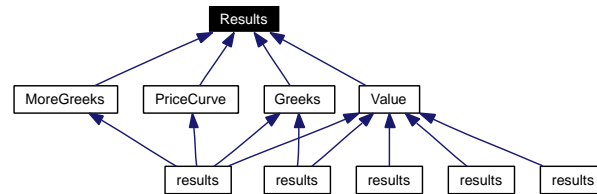
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

## 7.484 Results Class Reference

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

Inheritance diagram for Results:



### 7.484.1 Detailed Description

base class for generic result groups

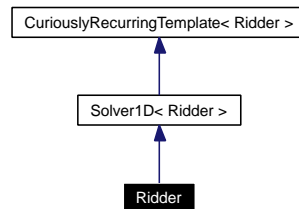
#### Public Member Functions

- virtual void **reset** ()=0

## 7.485 Ridder Class Reference

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

Inheritance diagram for Ridder:



### 7.485.1 Detailed Description

Ridder 1-D solver

#### Tests

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

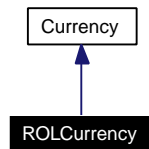
### Public Member Functions

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

## 7.486 ROLCurrency Class Reference

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

Inheritance diagram for ROLCurrency:



### 7.486.1 Detailed Description

Romanian leu.

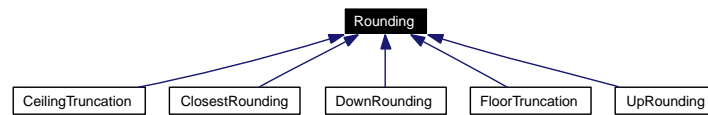
The ISO three-letter code is ROL; the numeric code is 642. It is divided in 100 bani.



## 7.487 Rounding Class Reference

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

Inheritance diagram for Rounding:



### 7.487.1 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*

### 7.487.2 Member Enumeration Documentation

#### 7.487.2.1 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

### 7.487.3 Constructor & Destructor Documentation

#### 7.487.3.1 [Rounding](#) ()

default constructor

Instances built through this constructor don't perform any rounding.

## 7.488 SalvagingAlgorithm Struct Reference

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

### 7.488.1 Detailed Description

algorithm used for matricial pseudo square root

### Public Types

- enum Type { None, Spectral, Hypersphere }

## 7.489 Sample Struct Template Reference

```
#include <ql/MonteCarlo/sample.hpp>
```

### 7.489.1 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

## 7.490 SampledCurve Class Reference

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

### 7.490.1 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)

## 7.490.2 Member Function Documentation

### 7.490.2.1 **Real** valueAtCenter () const

#### **Todo**

replace or complement with a more general function valueAt(spot)

### 7.490.2.2 **Real** firstDerivativeAtCenter () const

#### **Todo**

replace or complement with a more general function firstDerivativeAt(spot)

### 7.490.2.3 **Real** secondDerivativeAtCenter () const

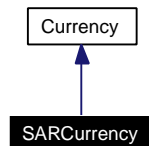
#### **Todo**

replace or complement with a more general function secondDerivativeAt(spot)

## 7.491 SARCurrency Class Reference

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

Inheritance diagram for SARCurrency:



### 7.491.1 Detailed Description

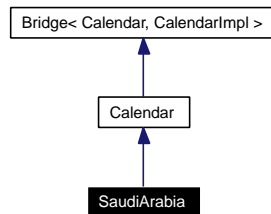
Saudi riyal.

The ISO three-letter code is SAR; the numeric code is 682. It is divided in 100 halalat.

## 7.492 SaudiArabia Class Reference

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

Inheritance diagram for SaudiArabia:



### 7.492.1 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



## 7.493 Schedule Class Reference

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

### 7.493.1 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 std::vector< [Date](#) > &, const [Calendar](#) &calendar=[NullCalendar](#)(), [BusinessDayConvention](#) convention=[Unadjusted](#))

### Date access

- [Size](#) **size** () const
- const [Date](#) & **operator[]** ([Size](#) i) const
- const [Date](#) & **date** ([Size](#) i) const
- const std::vector< [Date](#) > & **dates** () const
- bool **isRegular** ([Size](#) i) const

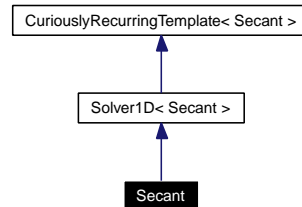
### Other inspectors

- const [Calendar](#) & **calendar** () const
- const [Date](#) & **startDate** () const
- const [Date](#) & **endDate** () const
- [Frequency](#) **frequency** () const
- [BusinessDayConvention](#) **businessDayConvention** () const

## 7.494 Secant Class Reference

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

Inheritance diagram for Secant:



### 7.494.1 Detailed Description

Secant 1-D solver

#### Tests

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

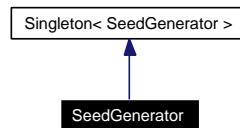
### Public Member Functions

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

## 7.495 SeedGenerator Class Reference

```
#include <ql/RandomNumbers/seedgenerator.hpp>
```

Inheritance diagram for SeedGenerator:



### 7.495.1 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>`

## 7.496 SegmentIntegral Class Reference

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

### 7.496.1 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.

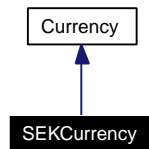
### Public Member Functions

- **SegmentIntegral** ([Size](#) intervals)
- **template<class F> Real operator()** (const F &f, [Real](#) a, [Real](#) b) const

## 7.497 SEKCurrency Class Reference

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

Inheritance diagram for SEKCurrency:



### 7.497.1 Detailed Description

Swedish krona.

The ISO three-letter code is SEK; the numeric code is 752. It is divided in 100 öre.

## 7.498 SequenceStatistics Class Template Reference

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

### 7.498.1 Detailed Description

```
template<class StatisticsType = Statistics> class QuantLib::SequenceStatistics< StatisticsType
>
```

Statistics analysis of N-dimensional (sequence) data.

It provides 1-dimensional statistics as discrepancy plus N-dimensional (sequence) statistics (e.g. mean, variance, skewness, kurtosis, etc.) with one component for each dimension of the sample space.

For most of the statistics this class relies on the StatisticsType underlying class to provide 1-D methods that will be iterated for all the components of the N-D data. These lifted methods are the union of all the methods that might be requested to the 1-D underlying StatisticsType class, with the usual compile-time checks provided by the template approach.

#### Tests

the correctness of the returned values is tested by checking them against numerical calculations.

### Public Types

- typedef StatisticsType **statistics\_type**

### Public Member Functions

- SequenceStatistics ([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)`

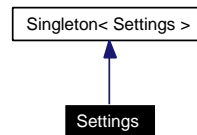
### Protected Attributes

- `Size dimension_`
- `std::vector< statistics_type > stats_`
- `std::vector< Real > results_`
- `Matrix quadraticSum_`

## 7.499 Settings Class Reference

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

Inheritance diagram for Settings:



### 7.499.1 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 &)`

### 7.499.2 Member Function Documentation

#### 7.499.2.1 `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.



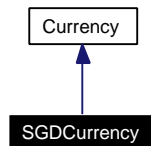
**Warning**

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.

## 7.500 SGDCurrency Class Reference

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

Inheritance diagram for SGDCurrency:



### 7.500.1 Detailed Description

[Singapore](#) dollar.

The ISO three-letter code is SGD; the numeric code is 702. It is divided in 100 cents.

## 7.501 Short Class Template Reference

```
#include <ql/CashFlows/shortindexedcoupon.hpp>
```

### 7.501.1 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

- **Short** ([Real](#) nominal, const [Date](#) &paymentDate, const boost::shared\_ptr< [Xibor](#) > &index, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())
- **Real amount** () const  
*inhibit calculation*

### 7.501.2 Member Function Documentation

#### 7.501.2.1 [Real amount](#) () const

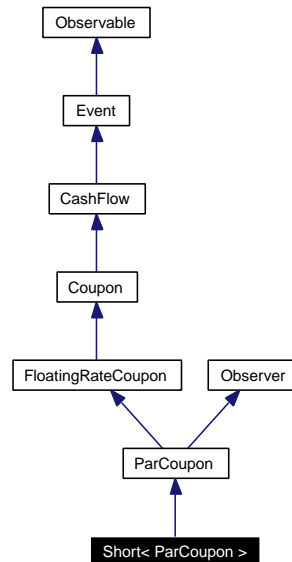
*inhibit calculation*

Unlike [ParCoupon](#), this coupon can't calculate its fixing for future dates, either.

## 7.502 Short< ParCoupon > Class Template Reference

```
#include <ql/CashFlows/shortfloatingcoupon.hpp>
```

Inheritance diagram for Short< ParCoupon >:



### 7.502.1 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** ([Real](#) nominal, const [Date](#) &paymentDate, const boost::shared\_ptr< [Xibor](#) > &index, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())
- **Real amount** () const  
*throws when an interpolated fixing is needed*

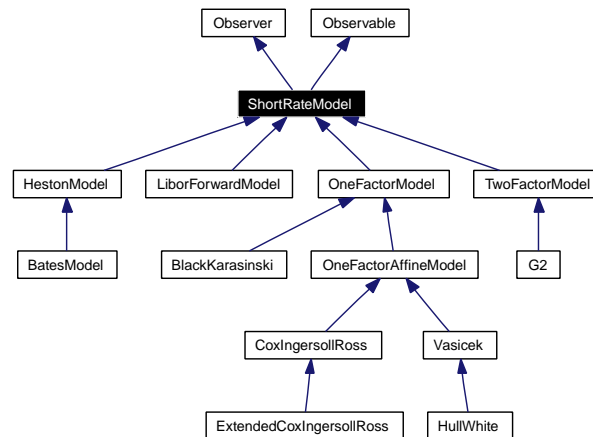
### Visitability

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

## 7.503 ShortRateModel Class Reference

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

Inheritance diagram for ShortRateModel:



### 7.503.1 Detailed Description

Abstract short-rate model class.

#### Public Member Functions

- **ShortRateModel** ([Size](#) nArguments)
- void [update](#) ()
- virtual boost::shared\_ptr< [NumericalMethod](#) > **tree** (const [TimeGrid](#) &) const =0
- 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`

## 7.503.2 Member Function Documentation

### 7.503.2.1 `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](#).

### 7.503.2.2 `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.

## 7.504 ShoutCondition Class Reference

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

### 7.504.1 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.

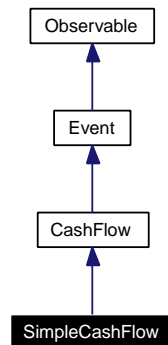
### Public Member Functions

- **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

## 7.505 SimpleCashFlow Class Reference

```
#include <ql/CashFlows/simplecashflow.hpp>
```

Inheritance diagram for SimpleCashFlow:



### 7.505.1 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

#### Visitability

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

### 7.505.2 Member Function Documentation

#### 7.505.2.1 [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](#).



7.505.2.2 [Date](#) date () const [virtual]

**Note:**

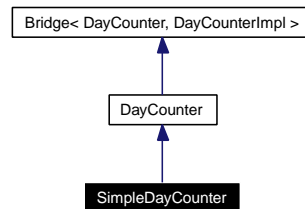
This is inheirited from the event class

Implements [CashFlow](#).

## 7.506 SimpleDayCounter Class Reference

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

Inheritance diagram for SimpleDayCounter:



### 7.506.1 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.

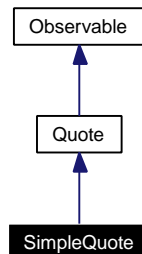
#### Tests

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

## 7.507 SimpleQuote Class Reference

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

Inheritance diagram for SimpleQuote:



### 7.507.1 Detailed Description

market element returning a stored value

**Examples:**

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

### Public Member Functions

- **SimpleQuote** ([Real](#) value)

#### Quote interface

- [Real](#) value () const  
*returns the current value*

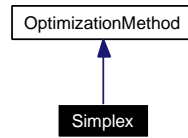
#### Modifiers

- void **setValue** ([Real](#) value)

## 7.508 Simplex Class Reference

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

Inheritance diagram for Simplex:



### 7.508.1 Detailed Description

Multi-dimensional simplex class.

#### Public Member Functions

- [Simplex](#) ([Real](#) lambda, [Real](#) tol)
- virtual void [minimize](#) (const [Problem](#) &P) const  
*minimize the optimization problem P*

### 7.508.2 Constructor & Destructor Documentation

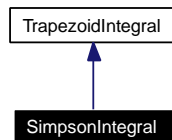
#### 7.508.2.1 [Simplex](#) ([Real](#) lambda, [Real](#) tol)

Constructor taking as input the characteristic length and tolerance

## 7.509 SimpsonIntegral Class Reference

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

Inheritance diagram for SimpsonIntegral:



### 7.509.1 Detailed Description

Integral of a one-dimensional function.

#### Tests

the correctness of the result is tested by checking it against known good values.

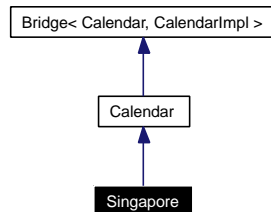
### Public Member Functions

- **SimpsonIntegral** ([Real](#) accuracy, [Size](#) maxIterations=[Null](#)< [Size](#) >())
- **template<class F> [Real](#) operator()** (const F &f, [Real](#) a, [Real](#) b) const

## 7.510 Singapore Class Reference

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

Inheritance diagram for Singapore:



### 7.510.1 Detailed Description

Singapore calendars

Holidays for the [Singapore](http://www.ses.com.sg) exchange (data from <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)

## 7.510.2 Member Enumeration Documentation

### 7.510.2.1 enum [Market](#)

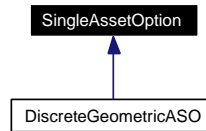
Enumerator:

SGX [Singapore](#) exchange.

## 7.511 SingleAssetOption Class Reference

```
#include <ql/Pricers/singleassetoption.hpp>
```

Inheritance diagram for SingleAssetOption:



### 7.511.1 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`

## 7.511.2 Member Function Documentation

7.511.2.1 [Volatility](#) `impliedVolatility (Real targetValue, Real accuracy = 1e-4, Size maxEvaluations = 100, Volatility minVol = QL_MIN_VOLATILITY, Volatility maxVol = QL_MAX_VOLATILITY) const`

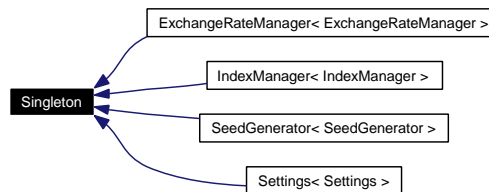
### [Warning](#)

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.

## 7.512 Singleton Class Template Reference

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

Inheritance diagram for Singleton:



### 7.512.1 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*

## 7.513 SingleVariate Struct Template Reference

```
#include <ql/MonteCarlo/mctraits.hpp>
```

### 7.513.1 Detailed Description

**template<class RNG = PseudoRandom> struct QuantLib::SingleVariate< RNG >**

default Monte Carlo traits for single-variate models

**Examples:**

[DiscreteHedging.cpp](#).

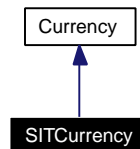
### Public Types

- 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**
- enum { **allowsErrorEstimate** = RNG::allowsErrorEstimate }

## 7.514 SITCurrency Class Reference

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

Inheritance diagram for SITCurrency:



### 7.514.1 Detailed Description

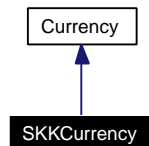
Slovenian tolar.

The ISO three-letter code is SIT; the numeric code is 705. It is divided in 100 stotinov.

## 7.515 SKKCurrency Class Reference

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

Inheritance diagram for SKKCurrency:



### 7.515.1 Detailed Description

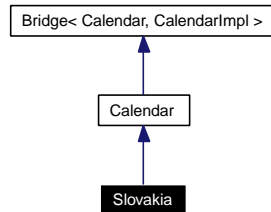
Slovak koruna.

The ISO three-letter code is SKK; the numeric code is 703. It is divided in 100 halierov.

## 7.516 Slovakia Class Reference

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

Inheritance diagram for Slovakia:



### 7.516.1 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)

## 7.516.2 Member Enumeration Documentation

### 7.516.2.1 enum [Market](#)

#### Enumerator:

*BSSE* Bratislava stock exchange.

## 7.517 SobolRsg Class Reference

```
#include <ql/RandomNumbers/sobolrsg.hpp>
```

### 7.517.1 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 primitivpolynomials.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

- typedef [Sample](#)< [Array](#) > **sample\_type**



- enum `DirectionIntegers` { `Unit`, `Jaeckel`, `SobolLevitan`, `SobolLevitanLemieux` }

## Public Member Functions

- `SobolRsg` (`Size` dimensionality, unsigned long seed=0, `DirectionIntegers` directionIntegers=`Jaeckel`)
- const std::vector< unsigned long > & `nextInt32Sequence` () const
- const `SobolRsg::sample_type` & `nextSequence` () const
- const `sample_type` & `lastSequence` () const
- `Size` `dimension` () const

## 7.517.2 Constructor & Destructor Documentation

- 7.517.2.1 `SobolRsg` (`Size` dimensionality, unsigned long seed = 0, `DirectionIntegers` directionIntegers = `Jaeckel`)

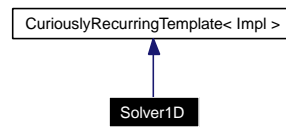
### Precondition:

dimensionality must be <= PPMT\_MAX\_DIM

## 7.518 Solver1D Class Template Reference

```
#include <ql/solver1d.hpp>
```

Inheritance diagram for Solver1D:



### 7.518.1 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\_

## 7.518.2 Member Function Documentation

### 7.518.2.1 [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 (i.e.,  $x$  is considered a zero if  $|f(x)| < accuracy$ ). This method contains a bracketing routine to which an initial guess must be supplied as well as a step used to scan the range of the possible bracketing values.

### 7.518.2.2 [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 (i.e.,  $x$  is considered a zero if  $|f(x)| < accuracy$ ). An initial guess must be supplied, as well as two values  $x_{\min}$  and  $x_{\max}$  which must bracket the zero (i.e., either  $f(x_{\min}) \leq 0 \leq f(x_{\max})$ , or  $f(x_{\max}) \leq 0 \leq f(x_{\min})$  must be true).

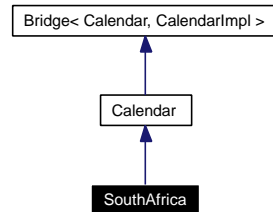
### 7.518.2.3 void setMaxEvaluations ([Size](#) *evaluations*)

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.

## 7.519 SouthAfrica Class Reference

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

Inheritance diagram for SouthAfrica:



### 7.519.1 Detailed Description

South-African calendar.

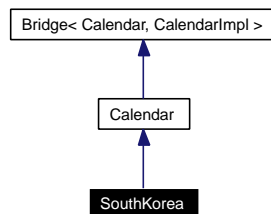
Holidays:

- 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)

## 7.520 SouthKorea Class Reference

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

Inheritance diagram for SouthKorea:



### 7.520.1 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)

## 7.520.2 Member Enumeration Documentation

### 7.520.2.1 enum [Market](#)

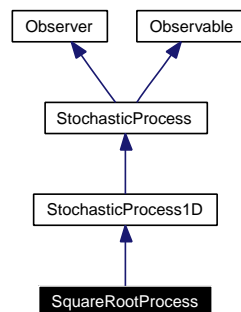
Enumerator:

*KRX* Korea exchange.

## 7.521 SquareRootProcess Class Reference

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

Inheritance diagram for SquareRootProcess:



### 7.521.1 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.  $\sigma(t, x_t)$*

## 7.522 StatsHolder Class Reference

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

### 7.522.1 Detailed Description

Helper class for precomputed distributions.

#### Public Member Functions

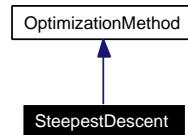
- **StatsHolder** ([Real](#) mean, [Real](#) standardDeviation)
- [Real](#) mean () const
- [Real](#) standardDeviation () const



## 7.523 SteepestDescent Class Reference

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

Inheritance diagram for SteepestDescent:



### 7.523.1 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*
- virtual [~SteepestDescent](#) ()  
*destructor*
- virtual void [minimize](#) (const [Problem](#) &P) const  
*minimize the optimization problem P*

## 7.524 `step_iterator` Class Template Reference

```
#include <ql/Utilities/steppingiterator.hpp>
```

### 7.524.1 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

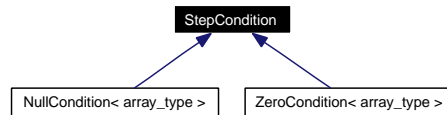
(Note that these are not member functions.)

- [step\\_iterator](#)< Iterator > [make\\_step\\_iterator](#) (Iterator it, [Size](#) step)  
*helper function to create step iterators*

## 7.525 StepCondition Class Template Reference

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

Inheritance diagram for StepCondition:



### 7.525.1 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

## 7.526 StepConditionSet Class Template Reference

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

### 7.526.1 Detailed Description

```
template<typename array_type> class QuantLib::StepConditionSet< array_type >
```

Parallel evolver for multiple arrays.

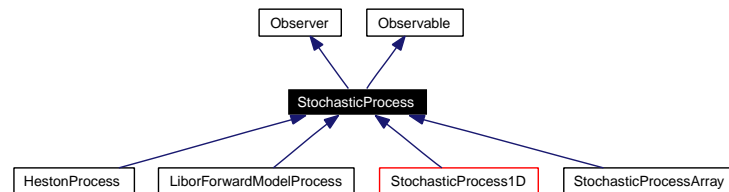
### Public Member Functions

- void **applyTo** (std::vector< array\_type > &a, [Time](#) t) const
- void **push\_back** (const itemType &a)

## 7.527 StochasticProcess Class Reference

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

Inheritance diagram for StochasticProcess:



### 7.527.1 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*

## 7.527.2 Member Function Documentation

**7.527.2.1** **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 [StochasticProcessArray](#).

**7.527.2.2** **virtual** [Disposable](#)<[Matrix](#)> **stdDeviation** (**Time** *t0*, const [Array](#) & *x0*, **Time** *dt*) const  
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 [StochasticProcessArray](#).

**7.527.2.3** **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 [LiborForwardModelProcess](#), and [StochasticProcessArray](#).

**7.527.2.4** **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](#).

**7.527.2.5** `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](#).

**7.527.2.6** `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 [BlackScholesProcess](#), [HestonProcess](#), [Merton76Process](#), and [StochasticProcessArray](#).

**7.527.2.7** `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.

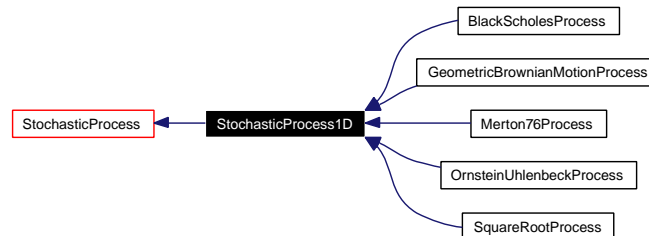
Implements [Observer](#).

Reimplemented in [BlackScholesProcess](#).

## 7.528 StochasticProcess1D Class Reference

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

Inheritance diagram for StochasticProcess1D:



### 7.528.1 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*

## 7.528.2 Member Function Documentation

### 7.528.2.1 virtual [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 in [OrnsteinUhlenbeckProcess](#).

### 7.528.2.2 virtual [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 in [OrnsteinUhlenbeckProcess](#).

### 7.528.2.3 virtual [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 in [OrnsteinUhlenbeckProcess](#).

### 7.528.2.4 virtual [Real](#) evolve ([Time](#) *t0*, [Real](#) *x0*, [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.

### 7.528.2.5 virtual [Real](#) apply ([Real](#) *x0*, [Real](#) *dx*) const [virtual]

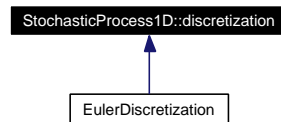
applies a change to the asset value. By default, it returns  $x + \Delta x$ .

Reimplemented in [BlackScholesProcess](#), and [Merton76Process](#).

## 7.529 StochasticProcess1D::discretization Class Reference

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

Inheritance diagram for StochasticProcess1D::discretization:



### 7.529.1 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

## 7.530 StochasticProcess::discretization Class Reference

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

Inheritance diagram for StochasticProcess::discretization:



### 7.530.1 Detailed Description

discretization of a stochastic process over a given time interval

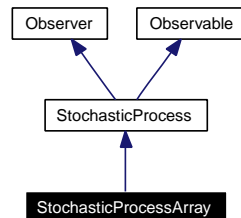
#### Public Member Functions

- 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

## 7.531 StochasticProcessArray Class Reference

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

Inheritance diagram for StochasticProcessArray:



### 7.531.1 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\_**

## 7.531.2 Member Function Documentation

**7.531.2.1** `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](#).

**7.531.2.2** `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](#).

**7.531.2.3** `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](#).

**7.531.2.4** `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](#).

**7.531.2.5** `Time time (const Date &) const` [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

**Note:**

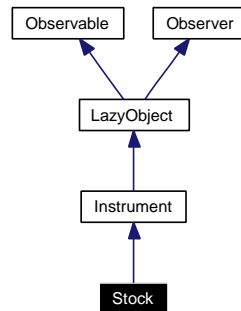
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

## 7.532 Stock Class Reference

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

Inheritance diagram for Stock:



### 7.532.1 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

### 7.532.2 Member Function Documentation

#### 7.532.2.1 void performCalculations () const [protected, virtual]

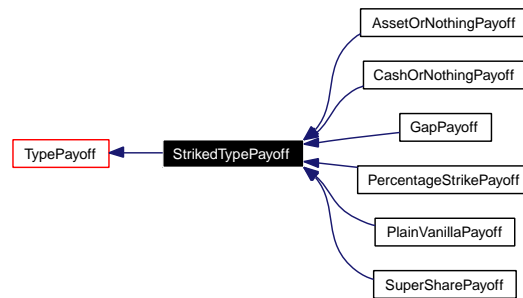
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

## 7.533 StrikedTypePayoff Class Reference

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

Inheritance diagram for StrikedTypePayoff:



### 7.533.1 Detailed Description

Intermediate class for payoffs based on a fixed strike.

#### Public Member Functions

- `StrikedTypePayoff` (Option::Type type, Real strike)
- `Real strike` () const

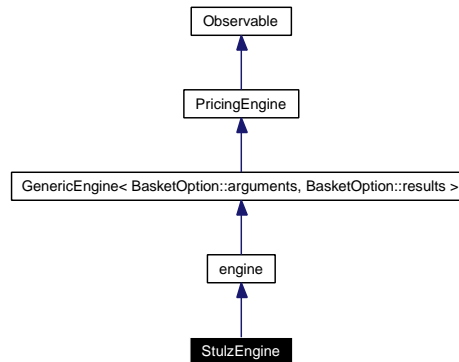
#### Protected Attributes

- `Real strike_`

## 7.534 StulzEngine Class Reference

```
#include <ql/PricingEngines/Basket/stulzengine.hpp>
```

Inheritance diagram for StulzEngine:



### 7.534.1 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

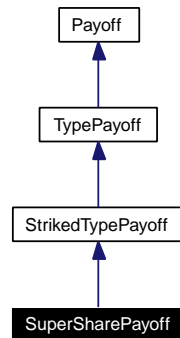
- `void calculate () const`



## 7.535 SuperSharePayoff Class Reference

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

Inheritance diagram for SuperSharePayoff:



### 7.535.1 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](#) &)

## 7.536 SVD Class Reference

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

### 7.536.1 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.

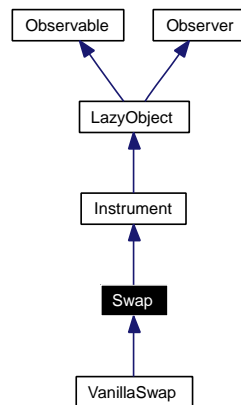
### Public Member Functions

- **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** ()

## 7.537 Swap Class Reference

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

Inheritance diagram for Swap:



### 7.537.1 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 std::vector< boost::shared\_ptr< [CashFlow](#) > > &firstLeg, const std::vector< boost::shared\_ptr< [CashFlow](#) > > &secondLeg, const [Handle](#)< [YieldTermStructure](#) > &termStructure)

#### Instrument interface

- bool [isExpired](#) () const  
*returns whether the instrument is still tradable.*

#### Additional interface

- [Date](#) [startDate](#) () const
- [Date](#) [maturity](#) () const
- [Real](#) [firstLegBPS](#) () const
- [Real](#) [secondLegBPS](#) () const
- [TimeBasket](#) [sensitivity](#) ([Integer](#) basis=2) const

### Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

## Protected Attributes

- `std::vector< boost::shared_ptr< CashFlow > > firstLeg_`
- `std::vector< boost::shared_ptr< CashFlow > > secondLeg_`
- `Handle< YieldTermStructure > termStructure_`
- `Real firstLegBPS_`
- `Real secondLegBPS_`

## 7.537.2 Member Function Documentation

### 7.537.2.1 [TimeBasket](#) sensitivity ([Integer](#) *basis* = 2) const

#### Bug

this method is not guaranteed to yield the right results.

#### Deprecated

this method will be removed in future releases.

### 7.537.2.2 `void setupExpired () const` [protected, virtual]

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

Reimplemented from [Instrument](#).

### 7.537.2.3 `void performCalculations () const` [protected, virtual]

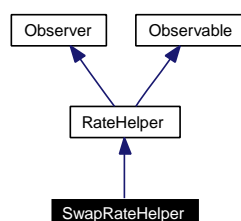
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

## 7.538 SwapRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for SwapRateHelper:



### 7.538.1 Detailed Description

Swap rate helper

#### Todo

currency and day counter of [Xibor](#) should be added to obtain well-defined [SwapRateHelper](#)

#### Warning

This class assumes that the settlement date does not change between calls of [setTermStructure\(\)](#).

#### Examples:

[swapvaluation.cpp](#).

### Public Member Functions

- [SwapRateHelper](#) (const [Handle](#)< [Quote](#) > &rate, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, [Frequency](#) floatingFrequency, [BusinessDayConvention](#) floatingConvention)
- [SwapRateHelper](#) ([Rate](#) rate, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, [Frequency](#) floatingFrequency, [BusinessDayConvention](#) floatingConvention)
- [SwapRateHelper](#) (const [Handle](#)< [Quote](#) > &rate, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, [Frequency](#) floatingFrequency, [BusinessDayConvention](#) floatingConvention, const [DayCounter](#) &floatingDayCount)
- [SwapRateHelper](#) ([Rate](#) rate, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, [Frequency](#) floatingFrequency, [BusinessDayConvention](#) floatingConvention, const [DayCounter](#) &floatingDayCount)
- [Real impliedQuote](#) () const
- [Date latestDate](#) () const  
*latest relevant date*

- void [setTermStructure](#) ([YieldTermStructure](#) \*)  
*sets the term structure to be used for pricing*

## Protected Attributes

- [Integer](#) [n\\_](#)
- [TimeUnit](#) [units\\_](#)
- [Integer](#) [settlementDays\\_](#)
- [Calendar](#) [calendar\\_](#)
- [BusinessDayConvention](#) [fixedConvention\\_](#)
- [BusinessDayConvention](#) [floatingConvention\\_](#)
- [Frequency](#) [fixedFrequency\\_](#)
- [Frequency](#) [floatingFrequency\\_](#)
- [DayCounter](#) [fixedDayCount\\_](#)
- [DayCounter](#) [floatingDayCount\\_](#)
- [Date](#) [settlement\\_](#)
- [Date](#) [latestDate\\_](#)
- [boost::shared\\_ptr](#)< [VanillaSwap](#) > [swap\\_](#)
- [Handle](#)< [YieldTermStructure](#) > [termStructureHandle\\_](#)

## 7.538.2 Constructor & Destructor Documentation

- 7.538.2.1 [SwapRateHelper](#) (const [Handle](#)< [Quote](#) > & *rate*, [Integer](#) *n*, [TimeUnit](#) *units*, [Integer](#) *settlementDays*, const [Calendar](#) & *calendar*, [Frequency](#) *fixedFrequency*, [BusinessDayConvention](#) *fixedConvention*, const [DayCounter](#) & *fixedDayCount*, [Frequency](#) *floatingFrequency*, [BusinessDayConvention](#) *floatingConvention*)

### Deprecated

use one of the other constructors

- 7.538.2.2 [SwapRateHelper](#) ([Rate](#) *rate*, [Integer](#) *n*, [TimeUnit](#) *units*, [Integer](#) *settlementDays*, const [Calendar](#) & *calendar*, [Frequency](#) *fixedFrequency*, [BusinessDayConvention](#) *fixedConvention*, const [DayCounter](#) & *fixedDayCount*, [Frequency](#) *floatingFrequency*, [BusinessDayConvention](#) *floatingConvention*)

### Deprecated

use one of the other constructors

## 7.538.3 Member Function Documentation

- 7.538.3.1 [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.

Implements [RateHelper](#).

### 7.538.3.2 void setTermStructure ([YieldTermStructure](#) \*) [virtual]

sets the term structure to be used for pricing

#### [Warning](#)

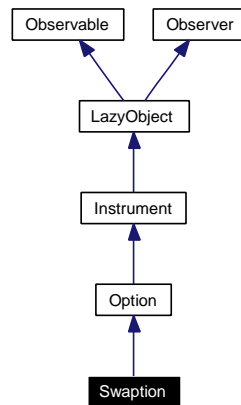
Being a pointer and not a `shared_ptr`, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

## 7.539 Swaption Class Reference

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

Inheritance diagram for Swaption:



### 7.539.1 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.

#### Todo

add 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)
- bool [isExpired](#) () const  
*returns whether the instrument is still tradable.*
- void [setupArguments](#) ([Arguments](#) \*) const



## Classes

- class [arguments](#)  
*Arguments for swaption calculation*
- class [results](#)  
*Results from swaption calculation*

## 7.539.2 Member Function Documentation

### 7.539.2.1 void setupArguments ([Arguments](#) \*) const [virtual]

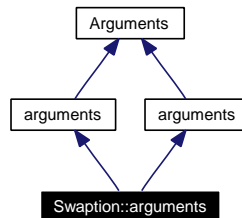
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

## 7.540 Swaption::arguments Class Reference

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

Inheritance diagram for Swaption::arguments:



### 7.540.1 Detailed Description

Arguments for swaption calculation

#### Public Member Functions

- `void validate () const`

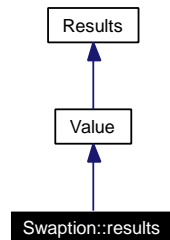
#### Public Attributes

- [Rate](#) `fairRate`
- [Rate](#) `fixedRate`
- [Real](#) `fixedBPS`

## 7.541 Swaption::results Class Reference

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

Inheritance diagram for Swaption::results:



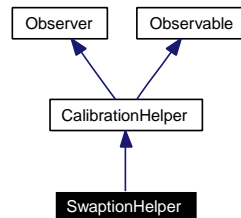
### 7.541.1 Detailed Description

Results from swaption calculation

## 7.542 SwaptionHelper Class Reference

```
#include <ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp>
```

Inheritance diagram for SwaptionHelper:



### 7.542.1 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 [Handle](#)< [YieldTermStructure](#) > &termStructure, bool calibrateVolatility=false)
- **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)
- 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.*

### 7.542.2 Constructor & Destructor Documentation

- 7.542.2.1 **SwaptionHelper** (const [Period](#) & maturity, const [Period](#) & length, const [Handle](#)< [Quote](#) > & volatility, const boost::shared\_ptr< [Xibor](#) > & index, [Frequency](#) fixedLegFrequency, const [DayCounter](#) & fixedLegDayCounter, const [Handle](#)< [YieldTermStructure](#) > & termStructure, bool calibrateVolatility = false)

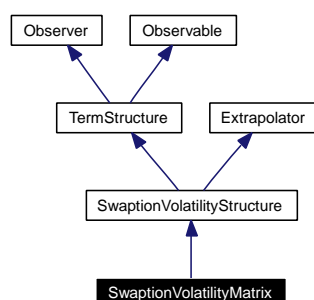
**Deprecated**

use the other constructor

## 7.543 SwaptionVolatilityMatrix Class Reference

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

Inheritance diagram for SwaptionVolatilityMatrix:



### 7.543.1 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.

#### Todo

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

### Public Member Functions

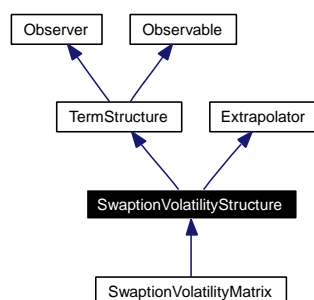
- **SwaptionVolatilityMatrix** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &exerciseDates, const std::vector< [Period](#) > &lengths, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter)
- [DayCounter](#) dayCounter () const  
*the day counter used for date/time conversion*
- const std::vector< [Date](#) > & exerciseDates () const
- const std::vector< [Period](#) > & lengths () const
- [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*

## 7.544 SwaptionVolatilityStructure Class Reference

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

Inheritance diagram for SwaptionVolatilityStructure:



### 7.544.1 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

#### Constructors

See the *TermStructure* documentation for issues regarding constructors.

- [SwaptionVolatilityStructure](#) ()  
*default constructor*
- [SwaptionVolatilityStructure](#) (const [Date](#) &referenceDate)  
*initialize with a fixed reference date*
- [SwaptionVolatilityStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)  
*calculate the reference date based on the global evaluation date*

#### Volatility

- [Volatility volatility](#) (const [Date](#) &start, const [Period](#) &length, [Rate](#) strike, bool extrapolate=false) const  
*returns the volatility for a given starting date and length*
- [Volatility volatility](#) ([Time](#) start, [Time](#) length, [Rate](#) strike, bool extrapolate=false) const  
*returns the volatility 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 [Time](#) `maxStartTime ()` const  
*the latest start time 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](#) `maxTimeLength ()` const  
*the largest length for which the term structure can return vols*
- 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 start, Time length, Rate strike)` const =0  
*implements the actual volatility calculation in derived classes*
- virtual `std::pair< Time, Time > convertDates (const Date &start, const Period &length)` const  
*implements the conversion between dates and times*

## 7.544.2 Constructor & Destructor Documentation

### 7.544.2.1 [SwaptionVolatilityStructure \(\)](#)

default constructor

#### **Warning**

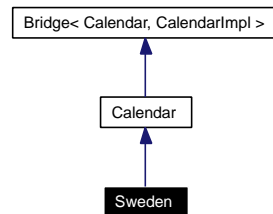
term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.



## 7.545 Sweden Class Reference

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

Inheritance diagram for Sweden:



### 7.545.1 Detailed Description

Swedish calendar.

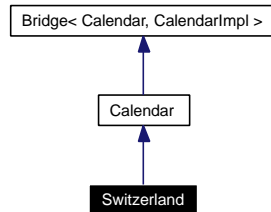
Holidays:

- 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

## 7.546 Switzerland Class Reference

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

Inheritance diagram for Switzerland:



### 7.546.1 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

## 7.547 SymmetricSchurDecomposition Class Reference

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

### 7.547.1 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

### 7.547.2 Constructor & Destructor Documentation

#### 7.547.2.1 [SymmetricSchurDecomposition](#) (const [Matrix](#) & s)

##### Precondition:

s must be symmetric

## 7.548 TabulatedGaussLegendre Class Reference

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

### 7.548.1 Detailed Description

tabulated Gauss-Legendre quadratures

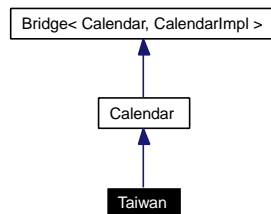
#### Public Member Functions

- **TabulatedGaussLegendre** ([Size](#) n=20)
- **template<class F> [Real](#) operator()** (const F &f) const
- **void order** ([Size](#))
- **[Size](#) order** () const

## 7.549 Taiwan Class Reference

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

Inheritance diagram for Taiwan:



### 7.549.1 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)

## 7.549.2 Member Enumeration Documentation

### 7.549.2.1 enum [Market](#)

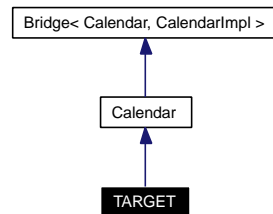
Enumerator:

*TSEC* [Taiwan](#) stock exchange.

## 7.550 TARGET Class Reference

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

Inheritance diagram for TARGET:



### 7.550.1 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.

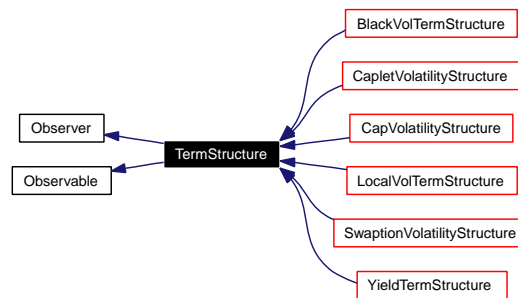
#### Examples:

[BermudanSwaption.cpp](#), [ConvertibleBonds.cpp](#), and [swapvaluation.cpp](#).

## 7.551 TermStructure Class Reference

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

Inheritance diagram for TermStructure:



### 7.551.1 Detailed Description

Basic term-structure functionality.

### Public Member Functions

#### Constructors

There are three ways in which a term structure can keep track of its reference date. The first is that such date is fixed; the second is that it is determined by advancing the current date of a given number of business days; and the third is that it is based on the reference date of some other structure.

In the first case, the constructor taking a date is to be used; the default implementation of `referenceDate()` will then return such date. In the second case, the constructor taking a number of days and a calendar is to be used; `referenceDate()` will return a date calculated based on the current evaluation date, and the term structure and its observers will be notified when the evaluation date changes. In the last case, the `referenceDate()` method must be overridden in derived classes so that it fetches and return the appropriate date.

- [TermStructure](#) ()  
default constructor
- [TermStructure](#) (const [Date](#) &referenceDate)  
initialize with a fixed reference date
- [TermStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)  
calculate the reference date based on the global evaluation date

#### Dates

- virtual const [Date](#) & [referenceDate](#) () const  
the reference date, i.e., the date at which discount = 1
- virtual [Calendar](#) [calendar](#) () const  
the calendar used for reference date calculation



- 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*

## 7.551.2 Constructor & Destructor Documentation

### 7.551.2.1 [TermStructure](#) ()

default constructor

#### Warning

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

## 7.551.3 Member Function Documentation

### 7.551.3.1 `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.

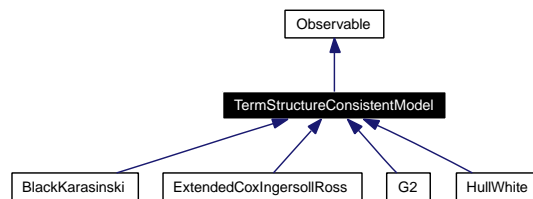
Implements [Observer](#).

Reimplemented in [AffineTermStructure](#), [ExtendedDiscountCurve](#), [FlatForward](#), and [CapVolatilityVector](#).

## 7.552 TermStructureConsistentModel Class Reference

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

Inheritance diagram for TermStructureConsistentModel:



### 7.552.1 Detailed Description

Term-structure consistent model class.

This is a base class for models that can reprice exactly any discount bond.

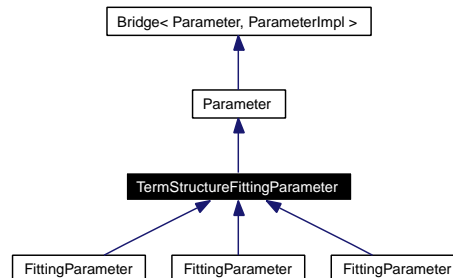
#### Public Member Functions

- `TermStructureConsistentModel` (const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- const [Handle](#)< [YieldTermStructure](#) > &termStructure () const

## 7.553 TermStructureFittingParameter Class Reference

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

Inheritance diagram for TermStructureFittingParameter:



### 7.553.1 Detailed Description

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

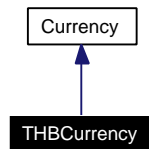
#### Public Member Functions

- `TermStructureFittingParameter` (const boost::shared\_ptr< Parameter::Impl > &impl)
- `TermStructureFittingParameter` (const [Handle](#)< [YieldTermStructure](#) > &term)

## 7.554 THBCurrency Class Reference

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

Inheritance diagram for THBCurrency:



### 7.554.1 Detailed Description

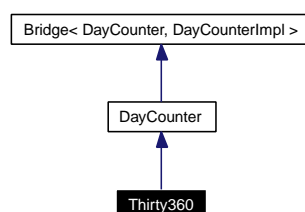
Thai baht.

The ISO three-letter code is THB; the numeric code is 764. It is divided in 100 stang.

## 7.555 Thirty360 Class Reference

```
#include <ql/DayCounters/thirty360.hpp>
```

Inheritance diagram for Thirty360:



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

### Public Types

- enum **Convention** {  
     **USA**, **BondBasis**, **European**, **EurobondBasis**,  
     **Italian** }

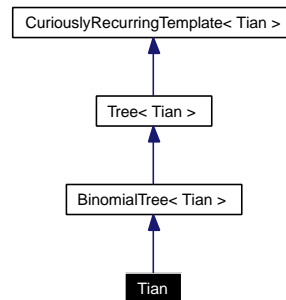
### Public Member Functions

- **Thirty360** (Convention c=Thirty360::USA)

## 7.556 Tian Class Reference

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

Inheritance diagram for Tian:



### 7.556.1 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

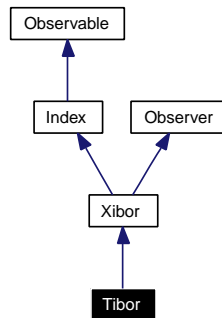
#### Protected Attributes

- [Real](#) up\_
- [Real](#) down\_
- [Real](#) pu\_
- [Real](#) pd\_

## 7.557 Tibor Class Reference

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

Inheritance diagram for Tibor:



### 7.557.1 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** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual365Fixed](#)())

## 7.558 TimeBasket Class Reference

```
#include <ql/CashFlows/timebasket.hpp>
```

### 7.558.1 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*



## 7.559 TimeGrid Class Reference

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

### 7.559.1 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
- [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

### Constructors

- [TimeGrid](#) ([Time](#) end, [Size](#) steps)  
*Regularly spaced time-grid.*
- template<class Iterator> [TimeGrid](#) (Iterator begin, Iterator end)  
*Time grid with mandatory time points.*
- template<class Iterator> [TimeGrid](#) (Iterator begin, Iterator end, [Size](#) steps)  
*Time grid with mandatory time points.*

### Time grid interface

- [Size](#) **findIndex** ([Time](#) t) const
- [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  $\text{grid}[i]$  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`

## 7.559.2 Constructor & Destructor Documentation

### 7.559.2.1 `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.

### 7.559.2.2 `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.

## 7.559.3 Member Function Documentation

### 7.559.3.1 `Size findIndex (Time t) const`

**Deprecated**

use `index()` instead

## 7.560 TqrEigenDecomposition Class Reference

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

### 7.560.1 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

## 7.561 TransformedGrid Class Reference

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

### 7.561.1 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

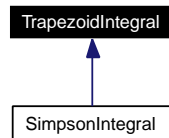
### Protected Attributes

- [Array](#) grid\_
- [Array](#) transformedGrid\_
- [Array](#) dxm\_
- [Array](#) dxp\_
- [Array](#) dx\_

## 7.562 TrapezoidIntegral Class Reference

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

Inheritance diagram for TrapezoidIntegral:



### 7.562.1 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

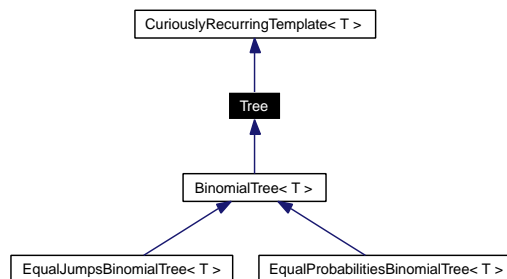
## Protected Attributes

- [Real](#) accuracy\_
- Method method\_
- [Size](#) maxIterations\_

## 7.563 Tree Class Template Reference

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

Inheritance diagram for Tree:



### 7.563.1 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...)

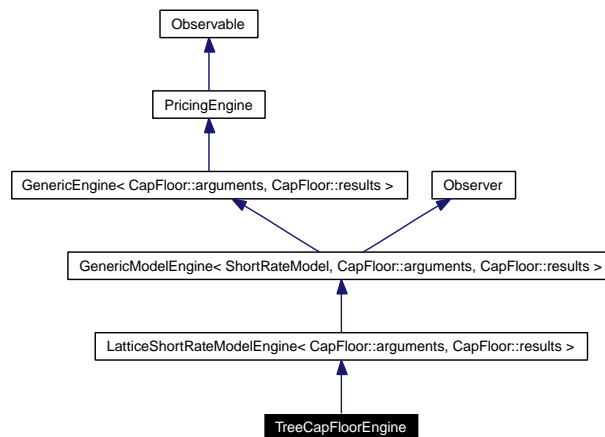
### Public Member Functions

- **Tree** ([Size](#) columns)
- [Size](#) columns () const

## 7.564 TreeCapFloorEngine Class Reference

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

Inheritance diagram for TreeCapFloorEngine:



### 7.564.1 Detailed Description

Numerical lattice engine for cap/floors.

#### Public Member Functions

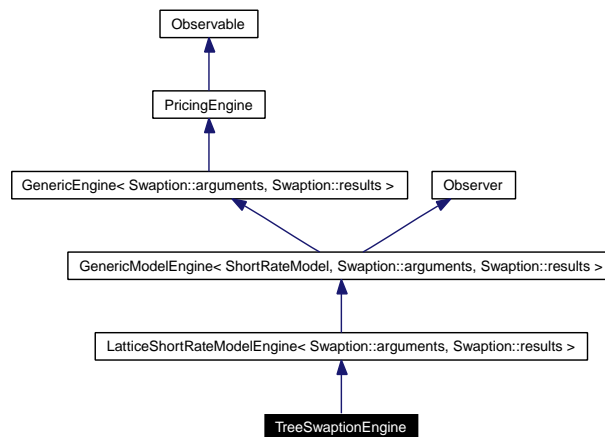
- **TreeCapFloorEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &model, [Size](#) timeSteps)
- **TreeCapFloorEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &model, const [TimeGrid](#) &timeGrid)
- void **calculate** () const



## 7.565 TreeSwaptionEngine Class Reference

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

Inheritance diagram for TreeSwaptionEngine:



### 7.565.1 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](#).

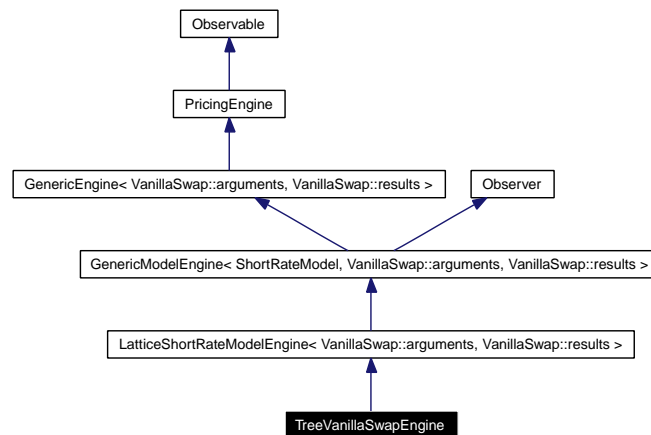
### Public Member Functions

- **TreeSwaptionEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &, [Size](#) timeSteps)
- **TreeSwaptionEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

## 7.566 TreeVanillaSwapEngine Class Reference

#include <ql/PricingEngines/Swaption/treeswaptionengine.hpp>

Inheritance diagram for TreeVanillaSwapEngine:



### 7.566.1 Detailed Description

Numerical lattice engine for simple swaps.

#### Tests

calculations are checked against known good results

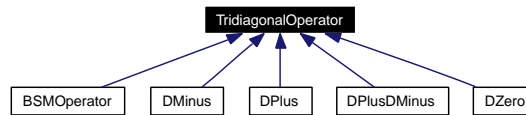
### Public Member Functions

- **TreeVanillaSwapEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &, [Size](#) timeSteps)
- **TreeVanillaSwapEngine** (const boost::shared\_ptr< [ShortRateModel](#) > &, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

## 7.567 TridiagonalOperator Class Reference

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

Inheritance diagram for TridiagonalOperator:



### 7.567.1 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*

## 7.568 TridiagonalOperator::TimeSetter Class Reference

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

### 7.568.1 Detailed Description

encapsulation of time-setting logic

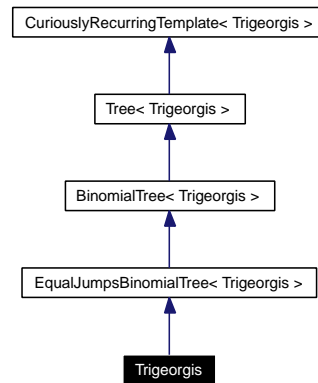
#### Public Member Functions

- virtual void **setTime** ([Time](#) t, [TridiagonalOperator](#) &L) const =0

## 7.569 Trigeorgis Class Reference

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

Inheritance diagram for Trigeorgis:



### 7.569.1 Detailed Description

Trigeorgis (additive equal jumps) binomial tree

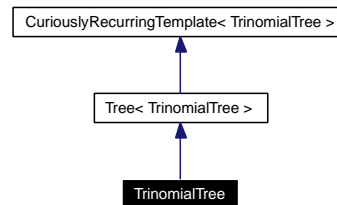
#### Public Member Functions

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

## 7.570 TrinomialTree Class Reference

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

Inheritance diagram for TrinomialTree:



### 7.570.1 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

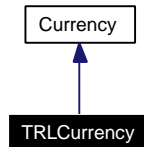
### Protected Attributes

- std::vector< [Branching](#) > **branchings\_**
- [Real](#) **x0\_**
- std::vector< [Real](#) > **dx\_**
- [TimeGrid](#) **timeGrid\_**

## 7.571 TRLCurrency Class Reference

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

Inheritance diagram for TRLCurrency:



### 7.571.1 Detailed Description

Turkish lira.

The ISO three-letter code is TRL; the numeric code is 792. It is divided in 100 kurus.

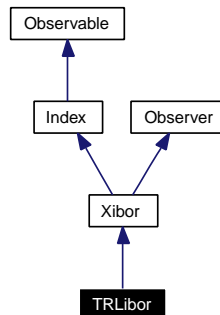
Obsoleted by the new Turkish lira since 2005.



## 7.572 TRLibor Class Reference

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

Inheritance diagram for TRLibor:



### 7.572.1 Detailed Description

TRY LIBOR rate

TRY LIBOR fixed by TBA.

See <<http://www.trlibor.org/trlibor/english/default.asp>>

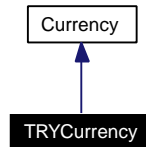
### Public Member Functions

- **TRLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.573 TRYCurrency Class Reference

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

Inheritance diagram for TRYCurrency:



### 7.573.1 Detailed Description

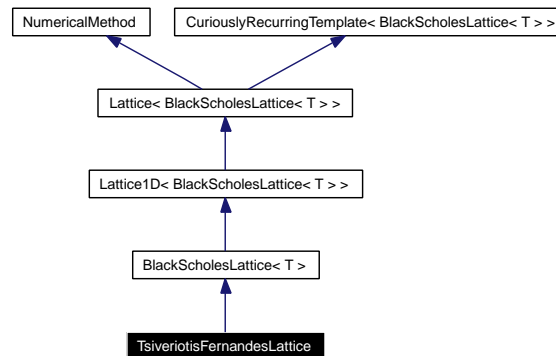
New Turkish lira.

The ISO three-letter code is TRY; the numeric code is 949. It is divided in 100 new kurus.

## 7.574 TsiveriotisFernandesLattice Class Template Reference

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

Inheritance diagram for TsiveriotisFernandesLattice:



### 7.574.1 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

### 7.574.2 Member Function Documentation

**7.574.2.1 void rollback** ([DiscretizedAsset](#) &, [Time](#) to) const [protected, virtual]

Roll back an asset until the given time, performing any needed adjustment.

Reimplemented from [Lattice< BlackScholesLattice< T > >](#).

**7.574.2.2 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:

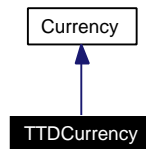
```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Reimplemented from [Lattice< BlackScholesLattice< T > >](#).

## 7.575 TTDCurrency Class Reference

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

Inheritance diagram for TTDCurrency:



### 7.575.1 Detailed Description

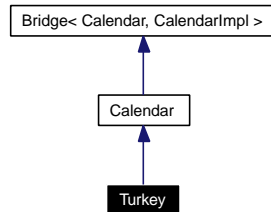
Trinidad & Tobago dollar.

The ISO three-letter code is TTD; the numeric code is 780. It is divided in 100 cents.

## 7.576 Turkey Class Reference

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

Inheritance diagram for Turkey:



### 7.576.1 Detailed Description

Turkish calendar.

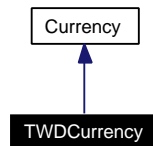
Holidays:

- 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)

## 7.577 TWDCurrency Class Reference

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

Inheritance diagram for TWDCurrency:



### 7.577.1 Detailed Description

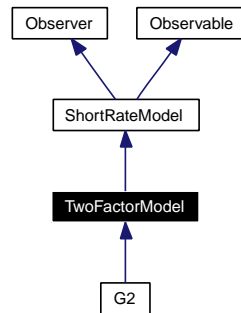
[Taiwan](#) dollar.

The ISO three-letter code is TWD; the numeric code is 901. It is divided in 100 cents.

## 7.578 TwoFactorModel Class Reference

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

Inheritance diagram for TwoFactorModel:



### 7.578.1 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.*



## 7.579 TwoFactorModel::ShortRateDynamics Class Reference

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

### 7.579.1 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

$$dW_t^x dW_t^y = \rho dt$$

.

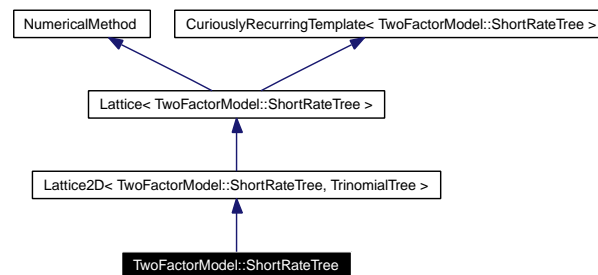
### Public Member Functions

- **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.*

## 7.580 TwoFactorModel::ShortRateTree Class Reference

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

Inheritance diagram for TwoFactorModel::ShortRateTree:



### 7.580.1 Detailed Description

Recombining two-dimensional tree discretizing the state variable.

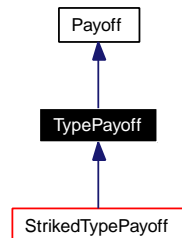
#### Public Member Functions

- [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

## 7.581 TypePayoff Class Reference

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

Inheritance diagram for TypePayoff:



### 7.581.1 Detailed Description

Intermediate class for call/put/straddle payoffs.

#### Public Member Functions

- **TypePayoff** (Option::Type type)
- Option::Type **optionType** () const

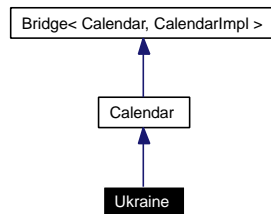
#### Protected Attributes

- Option::Type **type\_**

## 7.582 Ukraine Class Reference

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

Inheritance diagram for Ukraine:



### 7.582.1 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)

## 7.582.2 Member Enumeration Documentation

### 7.582.2.1 enum [Market](#)

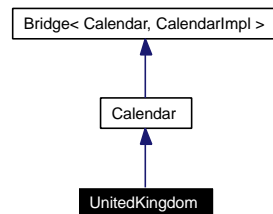
Enumerator:

*USE* Ukrainian stock exchange.

## 7.583 UnitedKingdom Class Reference

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

Inheritance diagram for UnitedKingdom:



### 7.583.1 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)

## 7.583.2 Member Enumeration Documentation

### 7.583.2.1 enum [Market](#)

UK calendars.

#### Enumerator:

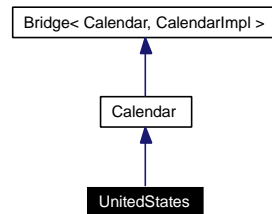
*Settlement* generic settlement calendar

*Exchange* London stock-exchange calendar.

## 7.584 UnitedStates Class Reference

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

Inheritance diagram for UnitedStates:



### 7.584.1 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)

### Tests

the correctness of the returned results is tested against a list of known holidays.

### Public Types

- enum `Market` { `Settlement`, `Exchange`, `NYSE`, `GovernmentBond` }
- US calendars.*

### Public Member Functions

- `UnitedStates` (`Market` market=`Settlement`)

## 7.584.2 Member Enumeration Documentation

### 7.584.2.1 enum [Market](#)

US calendars.

**Enumerator:**

*Settlement* generic settlement calendar

*Exchange* New York stock exchange calendar

**Deprecated**

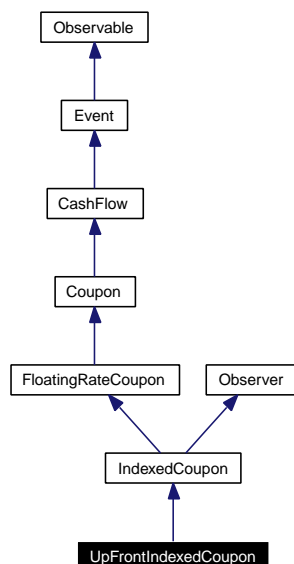
use NYSE instead

*NYSE* New York stock exchange calendar.

## 7.585 UpFrontIndexedCoupon Class Reference

```
#include <ql/CashFlows/upfrontindexedcoupon.hpp>
```

Inheritance diagram for UpFrontIndexedCoupon:



### 7.585.1 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** (*Real* nominal, const *Date* &paymentDate, const boost::shared\_ptr< *Xibor* > &index, const *Date* &startDate, const *Date* &endDate, *Integer* fixingDays, *Spread* spread=0.0, const *Date* &refPeriodStart=*Date*(), const *Date* &refPeriodEnd=*Date*(), const *DayCounter* &dayCounter=*DayCounter*())

#### FloatingRateCoupon interface

- *Date* fixingDate () const  
*fixing date*

#### Visitability

- virtual void accept (*AcyclicVisitor* &)

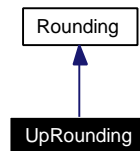
## Protected Attributes

- [Calendar](#) `calendar_`

## 7.586 UpRounding Class Reference

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

Inheritance diagram for UpRounding:



### 7.586.1 Detailed Description

Up-rounding.

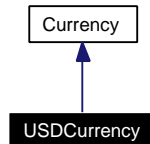
#### Public Member Functions

- **UpRounding** ([Integer](#) precision, [Integer](#) digit=5)

## 7.587 USDCurrency Class Reference

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

Inheritance diagram for USDCurrency:



### 7.587.1 Detailed Description

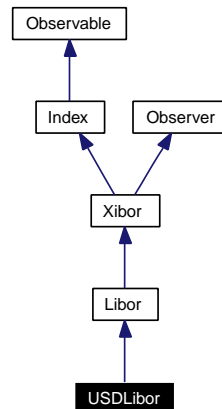
U.S. dollar.

The ISO three-letter code is USD; the numeric code is 840. It is divided in 100 cents.

## 7.588 USDLibor Class Reference

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

Inheritance diagram for USDLibor:



### 7.588.1 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>>.

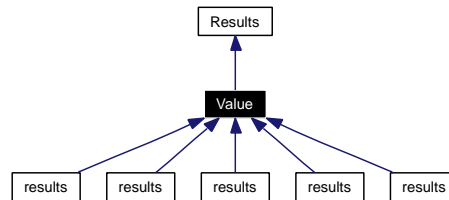
### Public Member Functions

- **USDLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## 7.589 Value Class Reference

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

Inheritance diagram for Value:



### 7.589.1 Detailed Description

pricing results

#### Public Member Functions

- `void reset ()`

#### Public Attributes

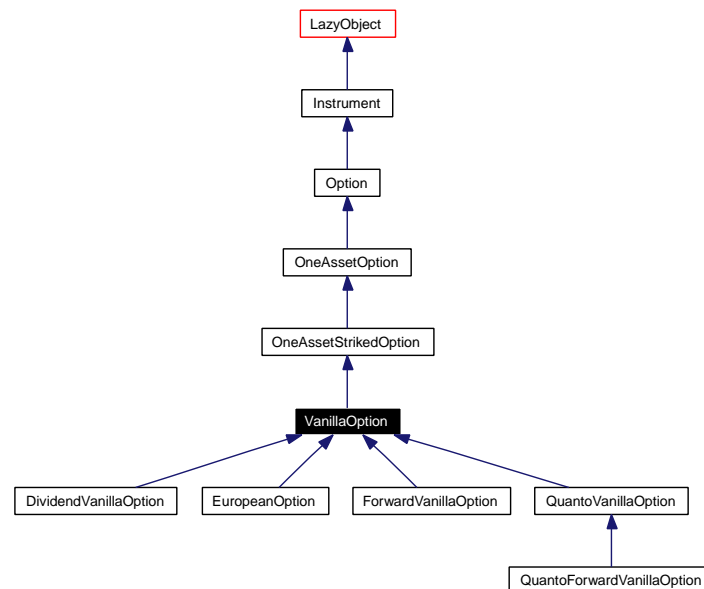
- `Real value`
- `Real errorEstimate`



## 7.590 VanillaOption Class Reference

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

Inheritance diagram for VanillaOption:



### 7.590.1 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](#) >())

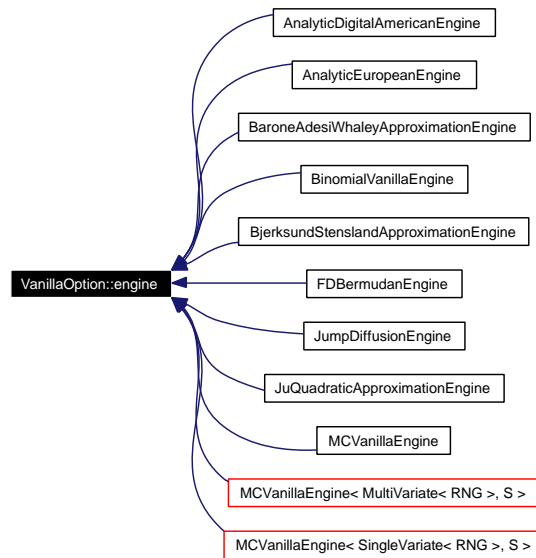
### Classes

- class [engine](#)  
*Vanilla option engine base class.*

## 7.591 VanillaOption::engine Class Reference

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

Inheritance diagram for VanillaOption::engine:



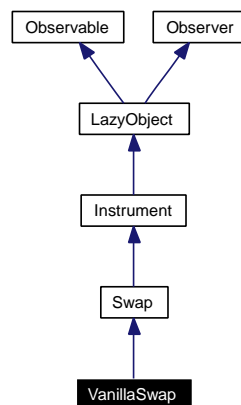
### 7.591.1 Detailed Description

Vanilla option engine base class.

## 7.592 VanillaSwap Class Reference

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

Inheritance diagram for VanillaSwap:



### 7.592.1 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 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, Integer indexFixingDays, 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*

## 7.592.2 Constructor & Destructor Documentation

- 7.592.2.1 [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` [Handle](#)< [YieldTermStructure](#) > & `termStructure`)

### Deprecated

use the other constructor

## 7.592.3 Member Function Documentation

- 7.592.3.1 `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](#).

- 7.592.3.2 `void` `fetchResults (const Results *)` const [virtual]

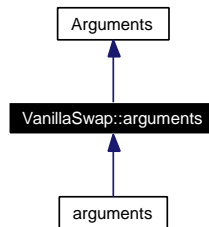
When a derived result structure is defined for an instrument, this method should be overridden to read from it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

## 7.593 VanillaSwap::arguments Class Reference

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

Inheritance diagram for VanillaSwap::arguments:



### 7.593.1 Detailed Description

Arguments for simple swap calculation

#### Public Member Functions

- void **validate** () const

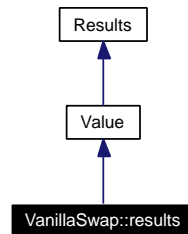
#### Public Attributes

- 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**

## 7.594 VanillaSwap::results Class Reference

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

Inheritance diagram for VanillaSwap::results:



### 7.594.1 Detailed Description

Results from simple swap calculation

#### Public Member Functions

- `void reset ()`

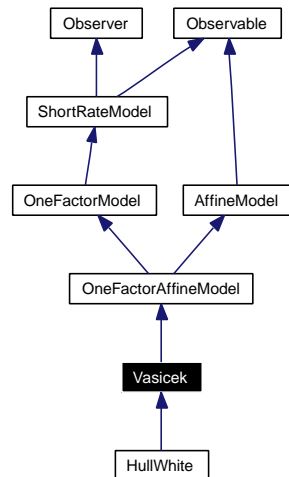
#### Public Attributes

- [Real](#) `fixedLegBPS`
- [Real](#) `floatingLegBPS`
- [Rate](#) `fairRate`
- [Spread](#) `fairSpread`

## 7.595 Vasicek Class Reference

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

Inheritance diagram for Vasicek:



### 7.595.1 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.*



## 7.596 Vasicek::Dynamics Class Reference

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

### 7.596.1 Detailed Description

Short-rate dynamics in the Vasicek model.

The short-rate follows an Ornstein-Uhlenbeck process with mean  $b$ .

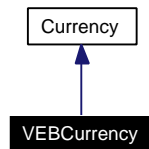
### Public Member Functions

- **Dynamics** ([Real](#) a, [Real](#) b, [Real](#) sigma, [Real](#) r0)
- virtual [Real](#) **variable** ([Time](#), [Rate](#) r) const
- virtual [Real](#) **shortRate** ([Time](#), [Real](#) x) const

## 7.597 VEBCurrency Class Reference

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

Inheritance diagram for VEBCurrency:



### 7.597.1 Detailed Description

Venezuelan bolivar.

The ISO three-letter code is VEB; the numeric code is 862. It is divided in 100 centimos.

## 7.598 Visitor Class Template Reference

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

### 7.598.1 Detailed Description

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

Visitor for a specific class

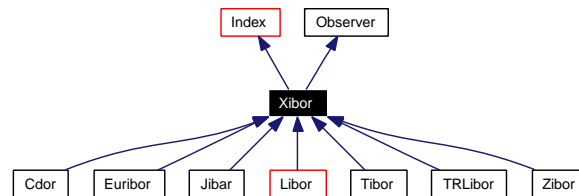
#### Public Member Functions

- virtual void **visit** (T &)=0

## 7.599 Xibor Class Reference

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

Inheritance diagram for Xibor:



### 7.599.1 Detailed Description

base class for LIBOR-like indexes

### Public Member Functions

- **Xibor** (const std::string &familyName, Integer n, TimeUnit units, Integer settlementDays, const Currency &currency, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter, const Handle< YieldTermStructure > &h)
- virtual Date valueDate (const Date &fixingDate) const
- virtual Date maturityDate (const Date &valueDate) const

#### Index interface

- Rate fixing (const Date &fixingDate) const  
*returns the fixing at the given date*

#### Observer interface

- void update ()

#### Inspectors

- std::string name () const  
*Returns the name of the index.*
- Period tenor () const
- Frequency frequency () const
- Integer settlementDays () const
- const Currency & currency () const
- Calendar calendar () const
- bool isAdjusted () const
- BusinessDayConvention businessDayConvention () const
- DayCounter dayCounter () const
- boost::shared\_ptr< YieldTermStructure > termStructure () const

## Protected Attributes

- `std::string` `familyName_`
- `Integer` `n_`
- `TimeUnit` `units_`
- `Integer` `settlementDays_`
- `Currency` `currency_`
- `Calendar` `calendar_`
- `BusinessDayConvention` `convention_`
- `DayCounter` `dayCounter_`
- `Handle`< `YieldTermStructure` > `termStructure_`

## 7.599.2 Member Function Documentation

### 7.599.2.1 `Rate` fixing (`const Date & fixingDate`) `const` [virtual]

returns the fixing at the given date

#### Note:

any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implements [Index](#).

### 7.599.2.2 `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](#).

### 7.599.2.3 `std::string name () const` [virtual]

Returns the name of the index.

#### Warning

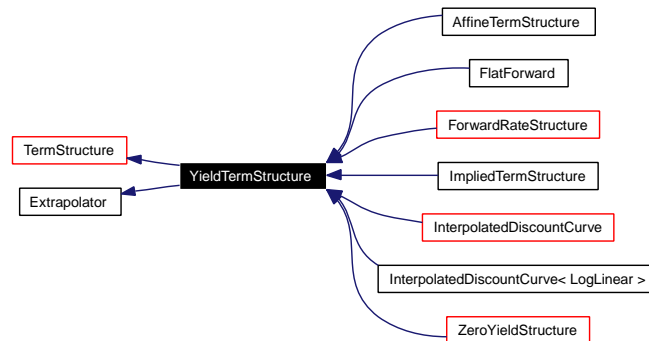
This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implements [Index](#).

## 7.600 YieldTermStructure Class Reference

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

Inheritance diagram for YieldTermStructure:



### 7.600.1 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

### Constructors

See the [TermStructure](#) documentation for issues regarding constructors.

- [YieldTermStructure](#) ()  
*default constructor*
- [YieldTermStructure](#) (const [Date](#) &referenceDate)  
*initialize with a fixed reference date*
- [YieldTermStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)  
*calculate the reference date based on the global evaluation date*

### 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` &resultDayCounter, 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

### Dates

- virtual `Date maxDate` () const =0  
the latest date for which the curve can return rates
- virtual `Time maxTime` () const  
the latest time for which the curve can return rates

## Protected Member Functions

### Calculations

These methods must be implemented in derived classes to perform the actual discount and rate calculations. 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*

## 7.600.2 Constructor & Destructor Documentation

### 7.600.2.1 [YieldTermStructure](#) ()

default constructor

#### Warning

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

## 7.600.3 Member Function Documentation

### 7.600.3.1 [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.

### 7.600.3.2 [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*.

### 7.600.3.3 [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*.

### 7.600.3.4 [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.

### 7.600.3.5 [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*.

### 7.600.3.6 [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.



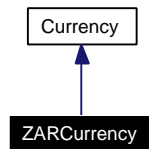
7.600.3.7 **Rate** parRate (const std::vector< **Time** > & *times*, **Frequency** *freq* = Annual, bool *extrapolate* = false) const

the first time in the vector must equal the start time; the following times must equal the payment times.

## 7.601 ZARCurrency Class Reference

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

Inheritance diagram for ZARCurrency:



### 7.601.1 Detailed Description

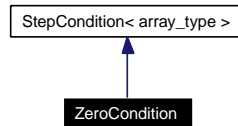
South-African rand.

The ISO three-letter code is ZAR; the numeric code is 710. It is divided into 100 cents.

## 7.602 ZeroCondition Class Template Reference

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

Inheritance diagram for ZeroCondition:



### 7.602.1 Detailed Description

```
template<class array_type> class QuantLib::ZeroCondition< array_type >
```

Zero exercise condition.

Used in CEV models

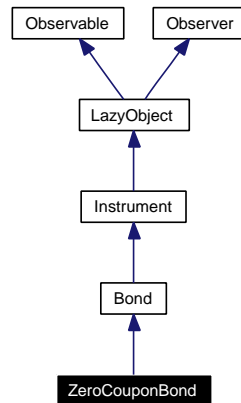
#### Public Member Functions

- void **applyTo** (array\_type &a, [Time](#)) const

## 7.603 ZeroCouponBond Class Reference

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

Inheritance diagram for ZeroCouponBond:



### 7.603.1 Detailed Description

zero-coupon bond

#### Tests

calculations are tested by checking results against cached values.

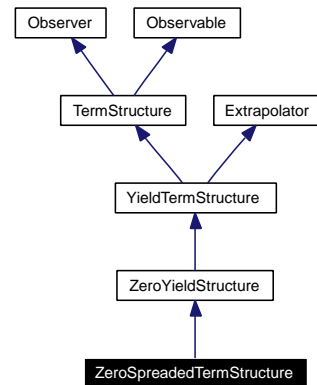
### Public Member Functions

- **ZeroCouponBond** (const [Date](#) &issueDate, const [Date](#) &maturityDate, [Integer](#) settlement-Days, const [DayCounter](#) &dayCounter, const [Calendar](#) &calendar, [BusinessDayConvention](#) paymentConvention=Following, [Real](#) redemption=100.0, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())

## 7.604 ZeroSpreadedTermStructure Class Reference

```
#include <ql/TermStructures/zerospreadedtermstructure.hpp>
```

Inheritance diagram for ZeroSpreadedTermStructure:



### 7.604.1 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 reference date, i.e., the date at which discount = 1*
- [Date](#) [maxDate](#) () const

*the latest date for which the curve can return rates*

- [Time maxTime](#) () const  
*the latest time for which the curve can return rates*

## Protected Member Functions

- [Rate zeroYieldImpl](#) (Time) const  
*returns the spreaded zero yield rate*
- [Rate forwardImpl](#) (Time) const  
*returns the spreaded forward rate*

## 7.605 ZeroYield Struct Reference

```
#include <ql/TermStructures/bootstraptraits.hpp>
```

### 7.605.1 Detailed Description

Zero-curve traits.

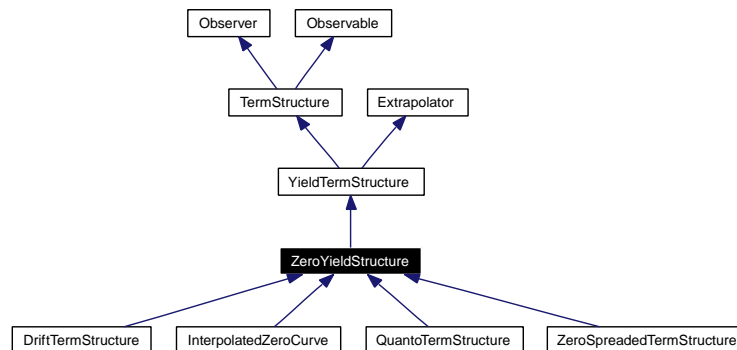
#### Static Public Member Functions

- 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)

## 7.606 ZeroYieldStructure Class Reference

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

Inheritance diagram for ZeroYieldStructure:



### 7.606.1 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.

### Public Member Functions

#### Constructors

See the [TermStructure](#) documentation for issues regarding constructors.

- [ZeroYieldStructure](#) ()
- [ZeroYieldStructure](#) (const [Date](#) &referenceDate)
- [ZeroYieldStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)

### Protected Member Functions

#### YieldTermStructure implementation

- [DiscountFactor](#) `discountImpl` ([Time](#)) const
- virtual [Rate](#) `zeroYieldImpl` ([Time](#)) const =0  
*zero-yield calculation*

### 7.606.2 Member Function Documentation

#### 7.606.2.1 [DiscountFactor](#) `discountImpl` ([Time](#)) const [protected, virtual]

Returns the discount factor for the given date calculating it from the zero yield.

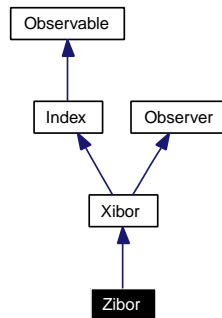


Implements [YieldTermStructure](#).

## 7.607 Zibor Class Reference

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

Inheritance diagram for Zibor:



### 7.607.1 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** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

## Chapter 8

# QuantLib File Documentation

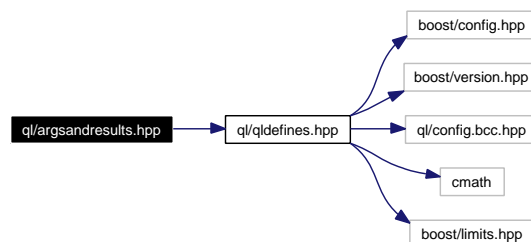
### 8.1 ql/argsandresults.hpp File Reference

#### 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

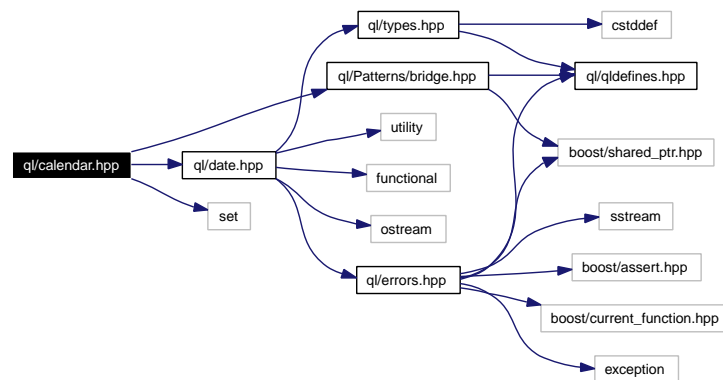
calendar class

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

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

```
#include <set>
```

Include 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 [QuantLib::BusinessDayConvention](#) {  
[QuantLib::Unadjusted](#), [QuantLib::Preceding](#), [QuantLib::ModifiedPreceding](#), [QuantLib::Following](#),

[QuantLib::ModifiedFollowing](#), [QuantLib::MonthEndReference](#) }

*Business Day conventions.*

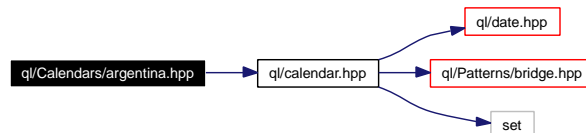
## 8.3 ql/Calendars/argentina.hpp File Reference

### 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/beijing.hpp File Reference

### 8.4.1 Detailed Description

Chinese calendar.

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

Include dependency graph for beijing.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **China**  
*Chinese calendar.*

### Typedefs

- typedef China **QuantLib::Beijing**



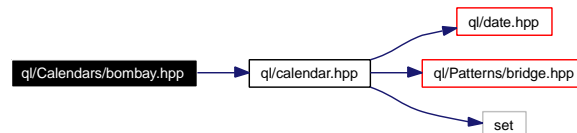
## 8.5 ql/Calendars/bombay.hpp File Reference

### 8.5.1 Detailed Description

Indian calendars.

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

Include dependency graph for `bombay.hpp`:



### Namespaces

- namespace `QuantLib`

### Classes

- class `India`  
*Indian calendars.*

### Typedefs

- typedef `India` `QuantLib::Bombay`

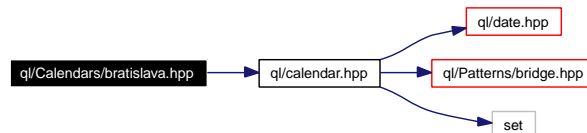
## 8.6 ql/Calendars/bratislava.hpp File Reference

### 8.6.1 Detailed Description

Slovak calendars.

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

Include dependency graph for bratislava.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Slovakia](#)  
*Slovak calendars.*

### Typedefs

- typedef Slovakia [QuantLib::Bratislava](#)

## 8.7 ql/Calendars/brazil.hpp File Reference

### 8.7.1 Detailed Description

Brazilian calendar.

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

Include dependency graph for brazil.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Brazil**  
*Brazilian calendar.*

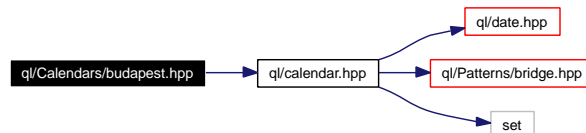
## 8.8 ql/Calendars/budapest.hpp File Reference

### 8.8.1 Detailed Description

Hungarian calendar.

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

Include dependency graph for budapest.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Hungary**  
*Hungarian calendar.*

### Typedefs

- typedef Hungary **QuantLib::Budapest**

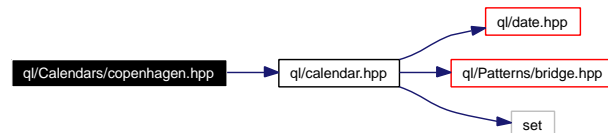
## 8.9 ql/Calendars/copenhagen.hpp File Reference

### 8.9.1 Detailed Description

Danish calendar.

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

Include dependency graph for copenhagen.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Denmark](#)  
*Danish calendar.*

### Typedefs

- typedef Denmark [QuantLib::Copenhagen](#)

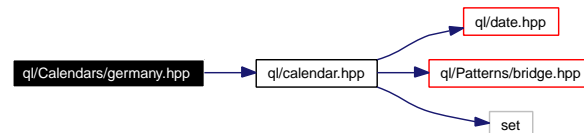
## 8.10 ql/Calendars/germany.hpp File Reference

### 8.10.1 Detailed Description

German calendars.

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

Include dependency graph for germany.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Germany](#)  
*German calendars.*

## 8.11 ql/Calendars/helsinki.hpp File Reference

### 8.11.1 Detailed Description

Finnish calendar.

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

Include dependency graph for helsinki.hpp:



### Namespaces

- namespace `QuantLib`

### Classes

- class `Finland`  
*Finnish calendar.*

### Typedefs

- typedef `Finland` `QuantLib::Helsinki`

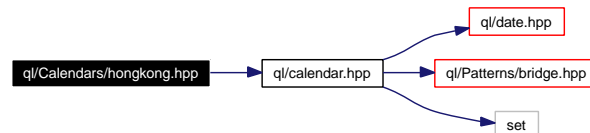
## 8.12 ql/Calendars/hongkong.hpp File Reference

### 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/iceland.hpp File Reference

### 8.13.1 Detailed Description

Icelandic calendars.

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

Include dependency graph for island.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Iceland](#)  
*Icelandic calendars.*

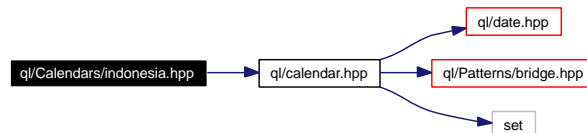
## 8.14 ql/Calendars/indonesia.hpp File Reference

### 8.14.1 Detailed Description

Indonesian calendars.

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

Include dependency graph for indonesia.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Indonesia](#)  
*Indonesian calendars*

## 8.15 ql/Calendars/istanbul.hpp File Reference

### 8.15.1 Detailed Description

Turkish calendar.

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

Include dependency graph for istanbul.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Turkey](#)  
*Turkish calendar.*

### Typedefs

- typedef Turkey [QuantLib::Istanbul](#)

## 8.16 ql/Calendars/italy.hpp File Reference

### 8.16.1 Detailed Description

Italian calendars.

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

Include dependency graph for italy.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Italy](#)  
*Italian calendars.*

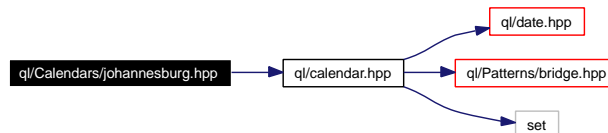
## 8.17 ql/Calendars/johannesburg.hpp File Reference

### 8.17.1 Detailed Description

South-African calendar.

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

Include dependency graph for johannesburg.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SouthAfrica](#)  
*South-African calendar.*

### Typedefs

- typedef SouthAfrica [QuantLib::Johannesburg](#)

## 8.18 ql/Calendars/jointcalendar.hpp File Reference

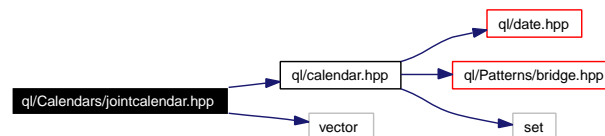
### 8.18.1 Detailed Description

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** { [QuantLib::JoinHolidays](#), [QuantLib::JoinBusinessDays](#) }  
*rules for joining calendars*

## 8.19 ql/Calendars/mexico.hpp File Reference

### 8.19.1 Detailed Description

Mexican calendars.

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

Include dependency graph for mexico.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Mexico**  
*Mexican calendars*

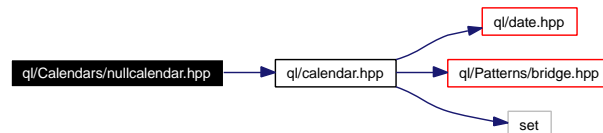
## 8.20 ql/Calendars/nullcalendar.hpp File Reference

### 8.20.1 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.*



## 8.21 ql/Calendars/oslo.hpp File Reference

### 8.21.1 Detailed Description

Norwegian calendar.

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

Include dependency graph for oslo.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Norway**  
*Norwegian calendar.*

### Typedefs

- typedef Norway **QuantLib::Oslo**

## 8.22 ql/Calendars/prague.hpp File Reference

### 8.22.1 Detailed Description

Czech calendars.

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

Include dependency graph for prague.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **CzechRepublic**  
*Czech calendars.*

### Typedefs

- typedef CzechRepublic **QuantLib::Prague**

## 8.23 ql/Calendars/riyadh.hpp File Reference

### 8.23.1 Detailed Description

Saudi Arabian calendar.

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

Include dependency graph for riyadh.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SaudiArabia](#)  
*Saudi Arabian calendar.*

### Typedefs

- typedef SaudiArabia [QuantLib::Riyadh](#)

## 8.24 ql/Calendars/seoul.hpp File Reference

### 8.24.1 Detailed Description

South Korean calendars.

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

Include dependency graph for seoul.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SouthKorea](#)  
*South Korean calendars.*

### Typedefs

- typedef [SouthKorea](#) [QuantLib::Seoul](#)

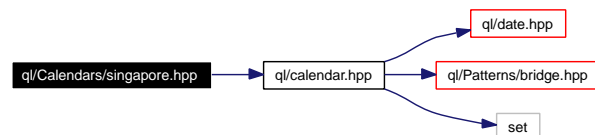
## 8.25 ql/Calendars/singapore.hpp File Reference

### 8.25.1 Detailed Description

Singapore calendars.

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

Include dependency graph for singapore.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Singapore](#)  
*Singapore calendars*

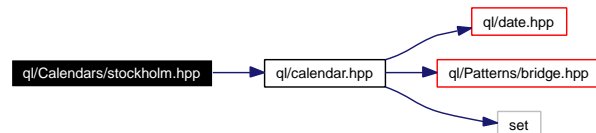
## 8.26 ql/Calendars/stockholm.hpp File Reference

### 8.26.1 Detailed Description

Swedish calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for stockholm.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Sweden](#)  
*Swedish calendar.*

### Typedefs

- typedef Sweden [QuantLib::Stockholm](#)

## 8.27 ql/Calendars/sydney.hpp File Reference

### 8.27.1 Detailed Description

Australian calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for sydney.hpp:



### Namespaces

- namespace `QuantLib`

### Classes

- class `Australia`  
*Australian calendar.*

### Typedefs

- typedef `Australia` `QuantLib::Sydney`

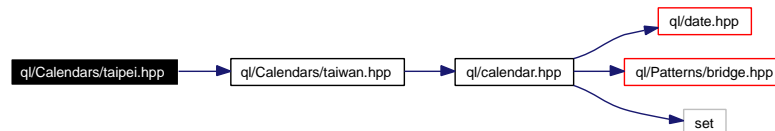
## 8.28 ql/Calendars/taipei.hpp File Reference

### 8.28.1 Detailed Description

Taipei calendar.

```
#include <ql/Calendars/taiwan.hpp>
```

Include dependency graph for taipei.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef Taiwan [QuantLib::Taipei](#)



## 8.29 ql/Calendars/taiwan.hpp File Reference

### 8.29.1 Detailed Description

Taiwanese calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for taiwan.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Taiwan](#)  
*Taiwanese calendars.*

## 8.30 ql/Calendars/target.hpp File Reference

### 8.30.1 Detailed Description

TARGET calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for target.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **TARGET**  
*TARGET calendar*

## 8.31 ql/Calendars/tokyo.hpp File Reference

### 8.31.1 Detailed Description

Japanese calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for tokyo.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Japan](#)  
*Japanese calendar.*

### Typedefs

- typedef Japan [QuantLib::Tokyo](#)

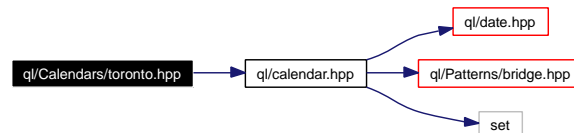
## 8.32 ql/Calendars/toronto.hpp File Reference

### 8.32.1 Detailed Description

Canadian calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for toronto.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Canada](#)  
*Canadian calendar.*

### Typedefs

- typedef [Canada](#) [QuantLib::Toronto](#)

## 8.33 ql/Calendars/ukraine.hpp File Reference

### 8.33.1 Detailed Description

Ukrainian calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for ukraine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Ukraine](#)  
*Ukrainian calendars.*

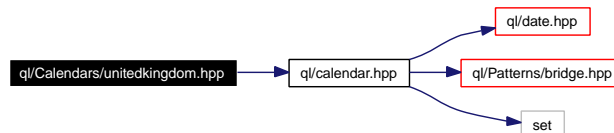
## 8.34 ql/Calendars/unitedkingdom.hpp File Reference

### 8.34.1 Detailed Description

UK calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for unitedkingdom.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [UnitedKingdom](#)  
*United Kingdom calendars.*

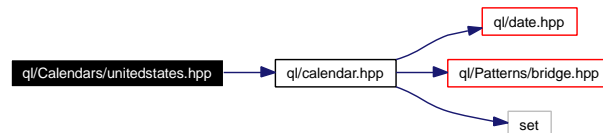
## 8.35 ql/Calendars/unitedstates.hpp File Reference

### 8.35.1 Detailed Description

US calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for unitedstates.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [UnitedStates](#)  
*United States calendars.*

## 8.36 ql/Calendars/warsaw.hpp File Reference

### 8.36.1 Detailed Description

Polish calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for warsaw.hpp:



### Namespaces

- namespace `QuantLib`

### Classes

- class `Poland`  
*Polish calendar.*

### Typedefs

- typedef `Poland` `QuantLib::Warsaw`



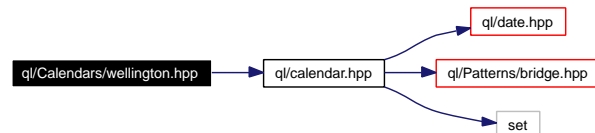
## 8.37 ql/Calendars/wellington.hpp File Reference

### 8.37.1 Detailed Description

New Zealand calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for wellington.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [NewZealand](#)  
*New Zealand calendar.*

### Typedefs

- typedef NewZealand [QuantLib::Wellington](#)

## 8.38 ql/Calendars/zurich.hpp File Reference

### 8.38.1 Detailed Description

Swiss calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for zurich.hpp:



### Namespaces

- namespace `QuantLib`

### Classes

- class `Switzerland`  
*Swiss calendar.*

### Typedefs

- typedef `Switzerland` `QuantLib::Zurich`

## 8.39 ql/capvolstructures.hpp File Reference

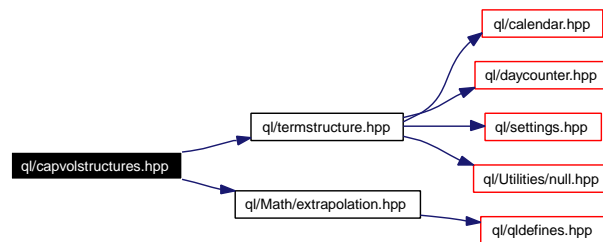
### 8.39.1 Detailed Description

Cap/Floor volatility structures.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/Math/extrapolation.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.*

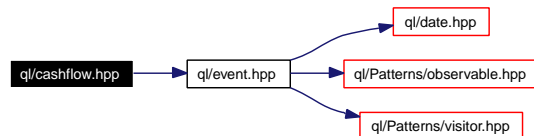
## 8.40 ql/cashflow.hpp File Reference

### 8.40.1 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.*

## 8.41 ql/CashFlows/analysis.hpp File Reference

### 8.41.1 Detailed Description

Cash-flow analysis functions.

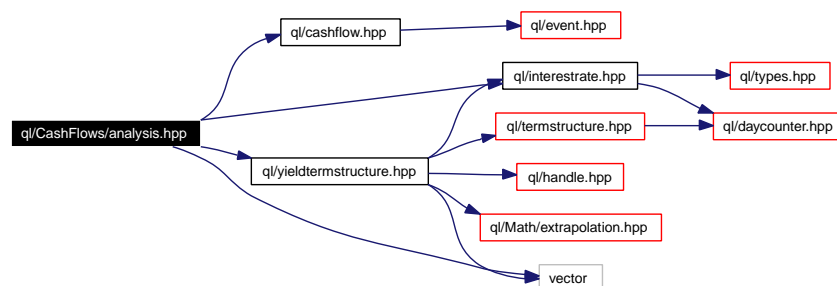
```
#include <ql/cashflow.hpp>
```

```
#include <ql/interestrate.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*

## 8.42 ql/CashFlows/basispointsensitivity.hpp File Reference

### 8.42.1 Detailed Description

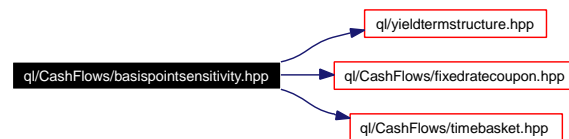
basis point sensitivity calculator

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/CashFlows/fixedratecoupon.hpp>
```

```
#include <ql/CashFlows/timebasket.hpp>
```

Include dependency graph for basispointsensitivity.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BPSBasketCalculator](#)

### Functions

- [Real](#) [QuantLib::BasisPointSensitivity](#) (const std::vector< boost::shared\_ptr< CashFlow > > &, const Handle< YieldTermStructure > &)  
*Collective basis-point sensitivity of a cash-flow sequence.*
- TimeBasket [QuantLib::BasisPointSensitivityBasket](#) (const std::vector< boost::shared\_ptr< CashFlow > > &, const Handle< YieldTermStructure > &, [Integer](#) basis)

## 8.43 ql/CashFlows/cashflowvectors.hpp File Reference

### 8.43.1 Detailed Description

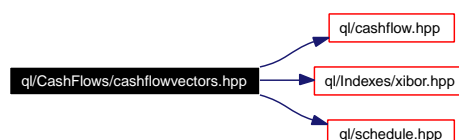
Cash flow vector builders.

```
#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 > > QuantLib::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 > > QuantLib::FloatingRateCouponVector` (const Schedule &schedule, [BusinessDayConvention](#) paymentAdjustment, const std::vector< [Real](#) > &nominals, const boost::shared\_ptr< Xibor > &index, [Integer](#) fixingDays, const std::vector< [Spread](#) > &spreads=std::vector< [Spread](#) >(), const DayCounter &dayCounter=DayCounter())  
*helper function building a sequence of par coupons*
- `std::vector< boost::shared_ptr< CashFlow > > QuantLib::DividendVector` (const std::vector< Date > &dividendDates, const std::vector< [Real](#) > &dividends)  
*helper function building a sequence of dividends*

## 8.44 ql/CashFlows/coupon.hpp File Reference

### 8.44.1 Detailed Description

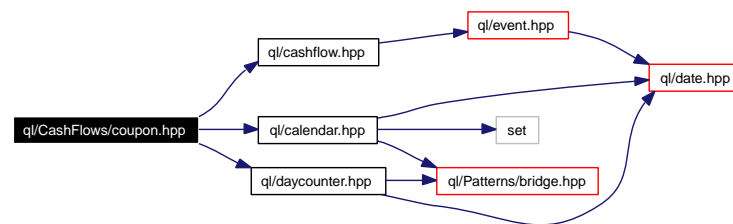
Coupon accruing over a fixed period.

```
#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*



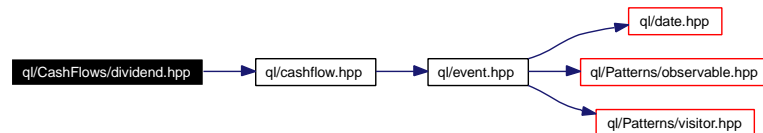
## 8.45 ql/CashFlows/dividend.hpp File Reference

### 8.45.1 Detailed Description

A stock dividend.

```
#include <ql/cashflow.hpp>
```

Include dependency graph for dividend.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Dividend**  
*Predetermined cash flow.*
- class **FixedDividend**  
*Predetermined cash flow.*
- class **FractionalDividend**  
*Predetermined cash flow.*

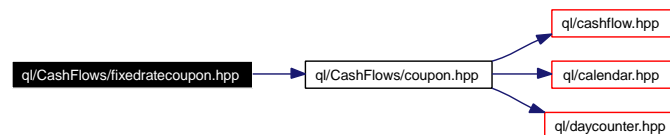
## 8.46 ql/CashFlows/fixedratecoupon.hpp File Reference

### 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

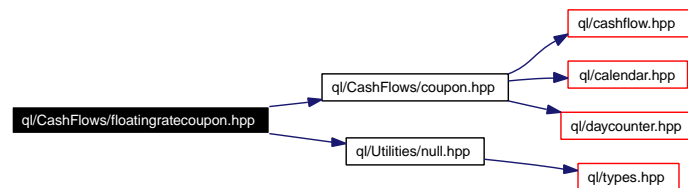
### 8.47.1 Detailed Description

Coupon paying a variable rate.

```
#include <ql/CashFlows/coupon.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

Include dependency graph for floatingratecoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **FloatingRateCoupon**  
*Coupon paying a variable rate*

## 8.48 ql/CashFlows/inarrearindexedcoupon.hpp File Reference

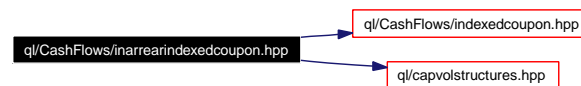
### 8.48.1 Detailed Description

in-arrear floating-rate coupon

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

```
#include <ql/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

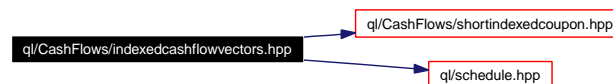
### 8.49.1 Detailed Description

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> std::vector< boost::shared_ptr< CashFlow > > QuantLib::IndexedCouponVector (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const boost::shared_ptr< Xibor > &index, Integer fixingDays, 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

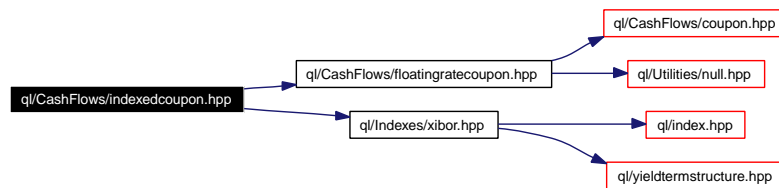
### 8.50.1 Detailed Description

indexed coupon

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for indexedcoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [IndexedCoupon](#)  
*Base indexed coupon class.*

## 8.51 ql/CashFlows/parcoupon.hpp File Reference

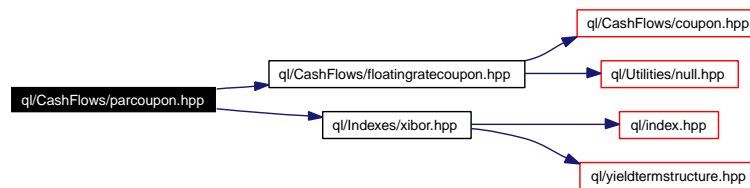
### 8.51.1 Detailed Description

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

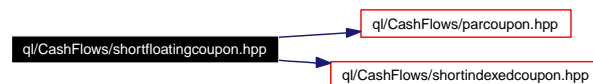
### 8.52.1 Detailed Description

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 dependency graph for shortindexedcoupon.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Short](#)  
*Short indexed coupon*

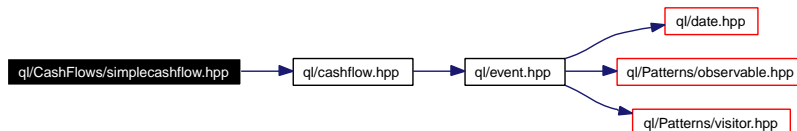
## 8.54 ql/CashFlows/simplecashflow.hpp File Reference

### 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

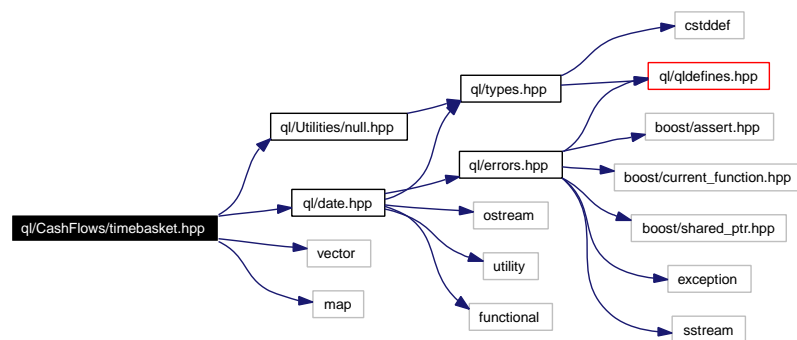
```
#include <ql/date.hpp>
```

```
#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

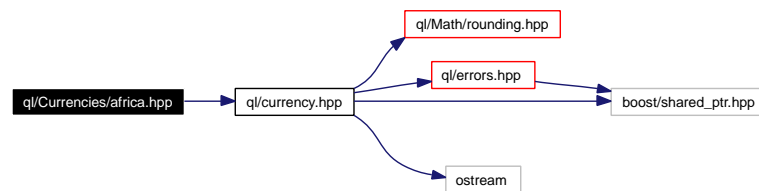
### 8.57.1 Detailed Description

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

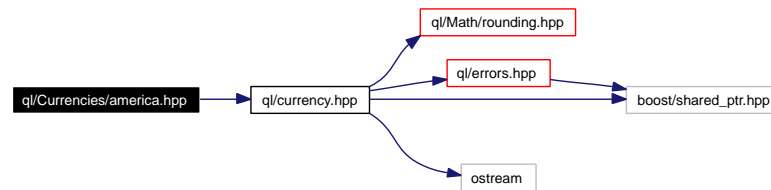
### 8.58.1 Detailed Description

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

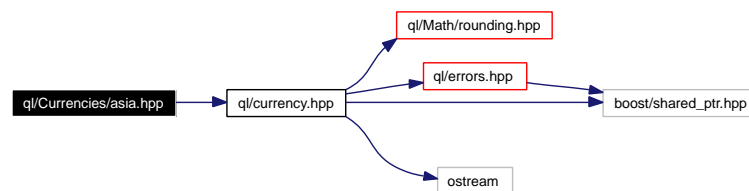
### 8.59.1 Detailed Description

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

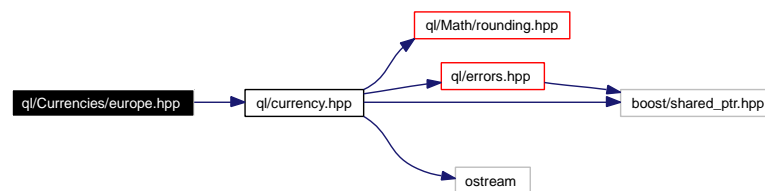
### 8.60.1 Detailed Description

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 [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>
```

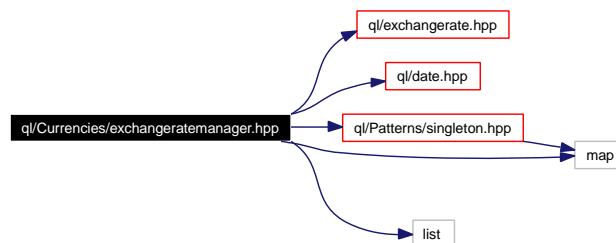
```
#include <ql/date.hpp>
```

```
#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

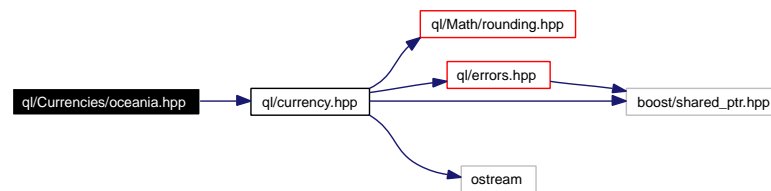
### 8.62.1 Detailed Description

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.

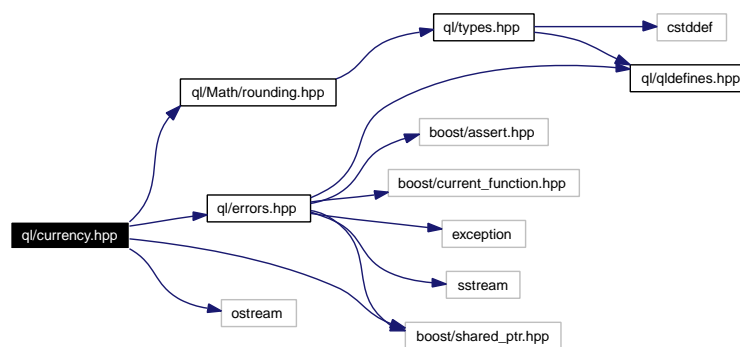
```
#include <ql/Math/rounding.hpp>
```

```
#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/errors.hpp>
```

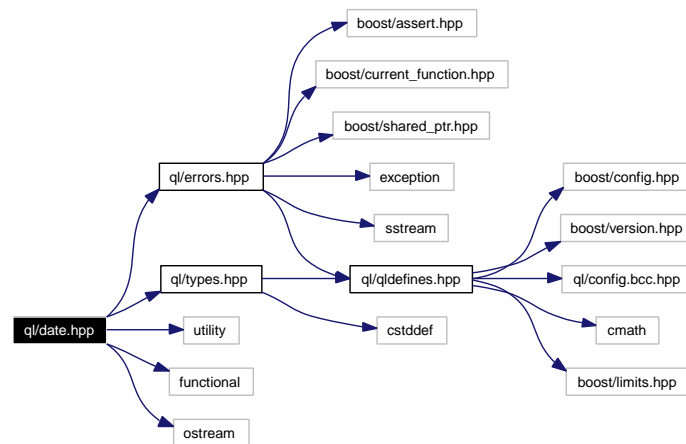
```
#include <ql/types.hpp>
```

```
#include <utility>
```

```
#include <functional>
```

```
#include <ostream>
```

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 **Period**  
*Time period described by a number of a given time unit.*
- class **Date**  
*Concrete date class.*



## Typedefs

- typedef [Integer QuantLib::Day](#)  
*Day number.*
- typedef [Integer QuantLib::Year](#)  
*Year number.*

## Enumerations

- enum [QuantLib::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 [QuantLib::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 [QuantLib::Frequency](#) {  
[QuantLib::NoFrequency](#) = -1, [QuantLib::Once](#) = 0, [QuantLib::Annual](#) = 1, [QuantLib::Semiannual](#) = 2,  
[QuantLib::EveryFourthMonth](#) = 3, [QuantLib::Quarterly](#) = 4, [QuantLib::Bimonthly](#) = 6,  
[QuantLib::Monthly](#) = 12 }  
*Frequency of events.*
- enum [QuantLib::TimeUnit](#) { **Days**, **Weeks**, **Months**, **Years** }  
*Units used to describe time periods.*

## Functions

- `std::ostream & QuantLib::detail::operator<< (std::ostream &, const long_weekday_holder &)`
- `std::ostream & QuantLib::detail::operator<< (std::ostream &, const short_weekday_holder &)`
- `std::ostream & QuantLib::detail::operator<< (std::ostream &, const shortest_weekday_holder &)`
- `detail::long_weekday_holder QuantLib::io::long\_weekday (Weekday)`  
*output weekdays in long format*

- detail::short\_weekday\_holder [QuantLib::io::short\\_weekday](#) ([Weekday](#))  
*output weekdays in short format (three letters)*
- detail::shortest\_weekday\_holder [QuantLib::io::shortest\\_weekday](#) ([Weekday](#))  
*output weekdays in shortest format (two letters)*
- std::ostream & **QuantLib::detail::operator<<** (std::ostream &, const long\_period\_holder &)
- std::ostream & **QuantLib::detail::operator<<** (std::ostream &, const short\_period\_holder &)
- detail::long\_period\_holder [QuantLib::io::long\\_period](#) (const Period &)  
*output periods in long format (e.g. "2 weeks")*
- detail::short\_period\_holder [QuantLib::io::short\\_period](#) (const Period &)  
*output periods in short format (e.g. "2w")*
- std::ostream & **QuantLib::detail::operator<<** (std::ostream &, const short\_date\_holder &)
- std::ostream & **QuantLib::detail::operator<<** (std::ostream &, const long\_date\_holder &)
- std::ostream & **QuantLib::detail::operator<<** (std::ostream &, const iso\_date\_holder &)
- detail::short\_date\_holder [QuantLib::io::short\\_date](#) (const Date &)  
*output dates in short format (mm/dd/yyyy)*
- detail::long\_date\_holder [QuantLib::io::long\\_date](#) (const Date &)  
*output dates in long format (Month ddth, yyyy)*
- detail::iso\_date\_holder [QuantLib::io::iso\\_date](#) (const Date &)  
*output dates in ISO format (yyyy-mm-dd)*
- Period **QuantLib::operator \*** ([Integer](#) n, [TimeUnit](#) units)
- Period **QuantLib::operator \*** ([TimeUnit](#) units, [Integer](#) n)
- bool **QuantLib::operator==** (const Period &p1, const Period &p2)
- bool **QuantLib::operator!=** (const Period &p1, const Period &p2)
- bool **QuantLib::operator>** (const Period &p1, const Period &p2)
- bool **QuantLib::operator<=** (const Period &p1, const Period &p2)
- bool **QuantLib::operator>=** (const Period &p1, const Period &p2)
- [BigInteger](#) **QuantLib::operator-** (const Date &d1, const Date &d2)
- bool **QuantLib::operator==** (const Date &d1, const Date &d2)
- bool **QuantLib::operator!=** (const Date &d1, const Date &d2)
- bool **QuantLib::operator<** (const Date &d1, const Date &d2)
- bool **QuantLib::operator<=** (const Date &d1, const Date &d2)
- bool **QuantLib::operator>** (const Date &d1, const Date &d2)
- bool **QuantLib::operator>=** (const Date &d1, const Date &d2)

## 8.65 ql/daycounter.hpp File Reference

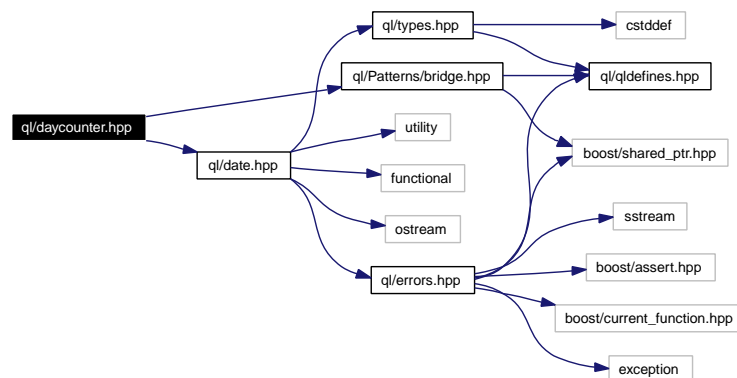
### 8.65.1 Detailed Description

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.*

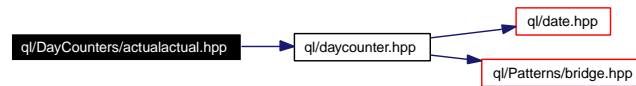
## 8.68 ql/DayCounters/actualactual.hpp File Reference

### 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.*

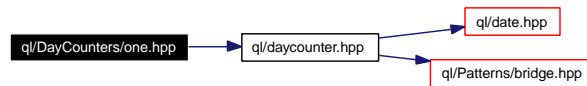
## 8.69 ql/DayCounters/one.hpp File Reference

### 8.69.1 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*

## 8.70 ql/DayCounters/simpliedaycounter.hpp File Reference

### 8.70.1 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.*



## 8.71 ql/DayCounters/thirty360.hpp File Reference

### 8.71.1 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*

## 8.72 ql/discretizedasset.hpp File Reference

### 8.72.1 Detailed Description

Discretized asset classes.

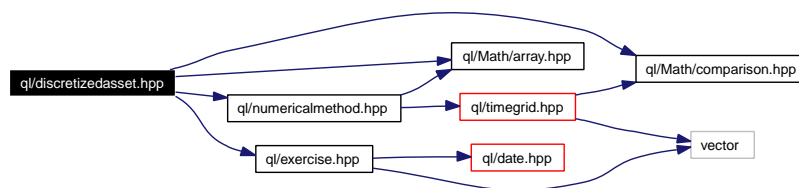
```
#include <ql/numericalmethod.hpp>
```

```
#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.*

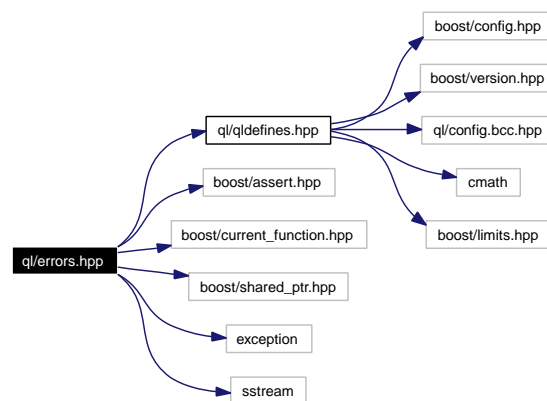
## 8.73 ql/errors.hpp File Reference

### 8.73.1 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*

## 8.73.2 Define Documentation

### 8.73.2.1 `#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)

### 8.73.2.2 `#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

### 8.73.2.3 `#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](#).

#### 8.73.2.4 #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

## 8.74 ql/event.hpp File Reference

### 8.74.1 Detailed Description

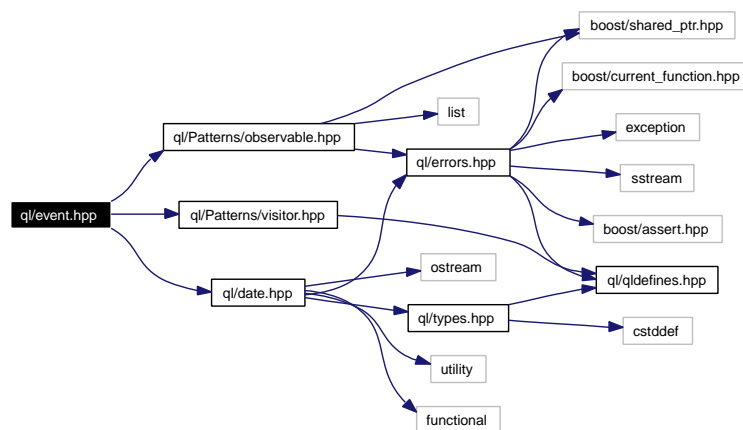
Base class for events associated with a given date.

```
#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.*

## 8.75 ql/exchangerate.hpp File Reference

### 8.75.1 Detailed Description

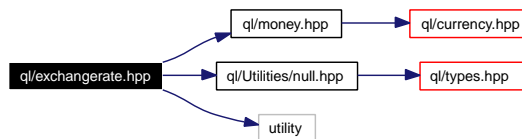
exchange rate between two currencies

```
#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*

## 8.76 ql/exercise.hpp File Reference

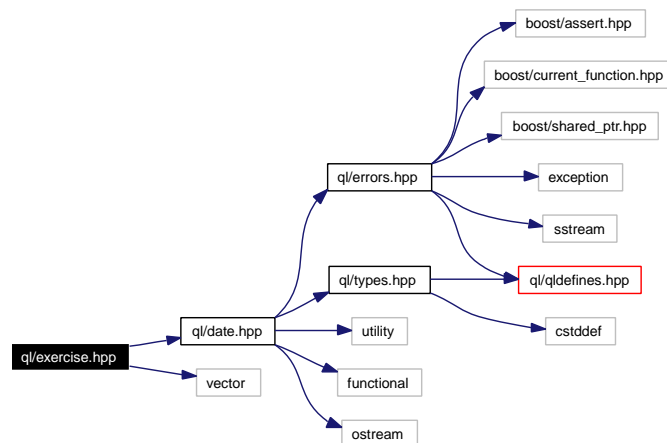
### 8.76.1 Detailed Description

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.*



## 8.77 ql/FiniteDifferences/americancondition.hpp File Reference

### 8.77.1 Detailed Description

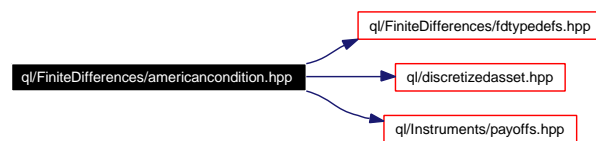
american option exercise condition

```
#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.*

## 8.78 ql/FiniteDifferences/boundarycondition.hpp File Reference

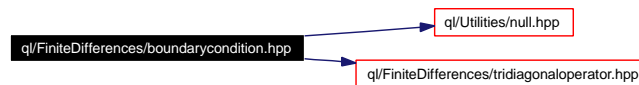
### 8.78.1 Detailed Description

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).*

## 8.79 ql/FiniteDifferences/bsmoperator.hpp File Reference

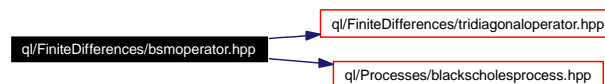
### 8.79.1 Detailed Description

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.*

## 8.80 ql/FiniteDifferences/bsmtermoperator.hpp File Reference

### 8.80.1 Detailed Description

differential operator for Black-Scholes-Merton equation

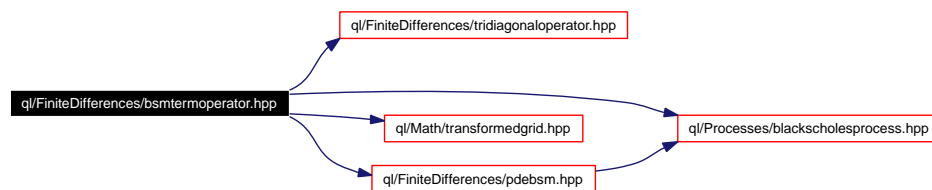
```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/Math/transformedgrid.hpp>
```

```
#include <ql/FiniteDifferences/pdebsm.hpp>
```

Include dependency graph for bsmtermoperator.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef PdeOperator< PdeBSM > [QuantLib::BSMTermOperator](#)  
*Black-Scholes-Merton differential operator.*

## 8.81 ql/FiniteDifferences/cranknicolson.hpp File Reference

### 8.81.1 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.*

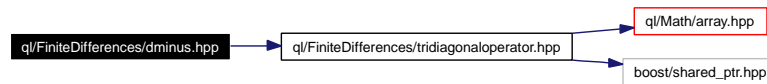
## 8.82 ql/FiniteDifferences/dminus.hpp File Reference

### 8.82.1 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*

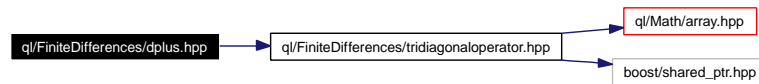
## 8.83 ql/FiniteDifferences/dplus.hpp File Reference

### 8.83.1 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*

## 8.84 ql/FiniteDifferences/dplusdminus.hpp File Reference

### 8.84.1 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*



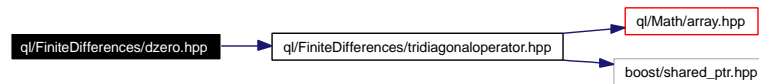
## 8.85 ql/FiniteDifferences/dzero.hpp File Reference

### 8.85.1 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*

## 8.86 ql/FiniteDifferences/expliciteuler.hpp File Reference

### 8.86.1 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.*

## 8.87 ql/FiniteDifferences/fdtypefs.hpp File Reference

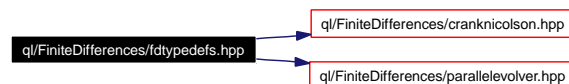
### 8.87.1 Detailed Description

default choices for template instantiations

```
#include <ql/FiniteDifferences/cranknicolson.hpp>
```

```
#include <ql/FiniteDifferences/parallelevolver.hpp>
```

Include dependency graph for fdtypefs.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `FiniteDifferenceModel< CrankNicolson< TridiagonalOperator > >` [QuantLib::StandardFiniteDifferenceModel](#)  
*default choice for finite-difference model*
- typedef `FiniteDifferenceModel< ParallelEvolver< CrankNicolson< TridiagonalOperator > >` [QuantLib::StandardSystemFiniteDifferenceModel](#)  
*default choice for parallel finite-difference model*
- typedef `StepCondition< Array >` [QuantLib::StandardStepCondition](#)  
*default choice for step condition*
- typedef `CurveDependentStepCondition< Array >` [QuantLib::StandardCurveDependent-StepCondition](#)

## 8.88 ql/FiniteDifferences/finitedifferencemodel.hpp File Reference

### 8.88.1 Detailed Description

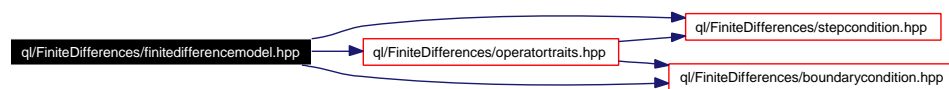
generic finite difference model

```
#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.*

## 8.89 ql/FiniteDifferences/impliciteuler.hpp File Reference

### 8.89.1 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.*

## 8.90 ql/FiniteDifferences/mixedscheme.hpp File Reference

### 8.90.1 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.*

## 8.91 ql/FiniteDifferences/onefactoroperator.hpp File Reference

### 8.91.1 Detailed Description

general differential operator for one-factor interest rate models

```
#include <ql/qldefines.hpp>
```

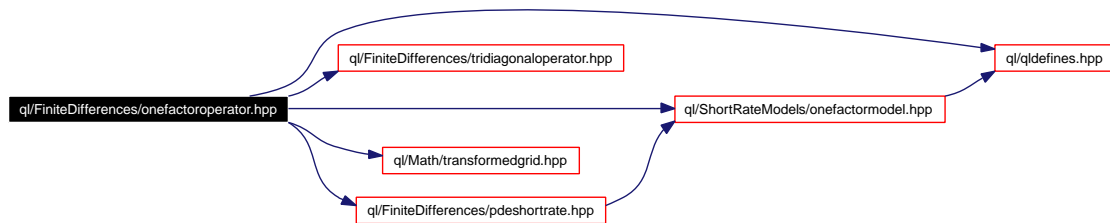
```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#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 >` [QuantLib::OneFactorOperator](#)  
*Interest-rate single factor model differential operator.*

## 8.92 ql/FiniteDifferences/operatorfactory.hpp File Reference

### 8.92.1 Detailed Description

Factory for finite difference operators.

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

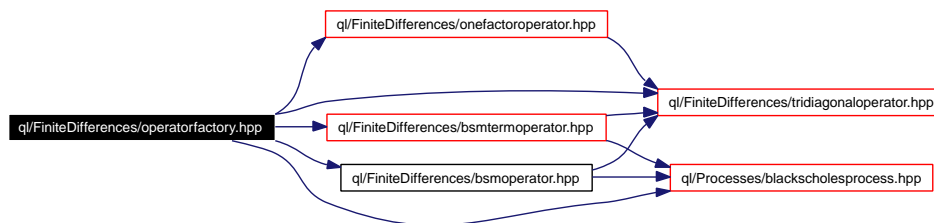
```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#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.*



## 8.93 ql/FiniteDifferences/operatortraits.hpp File Reference

### 8.93.1 Detailed Description

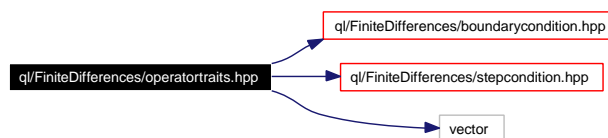
Differential operator traits.

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

```
#include <vector>
```

Include dependency graph for operatortraits.hpp:



### Namespaces

- namespace **QuantLib**

## 8.94 ql/FiniteDifferences/parallelevolver.hpp File Reference

### 8.94.1 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.

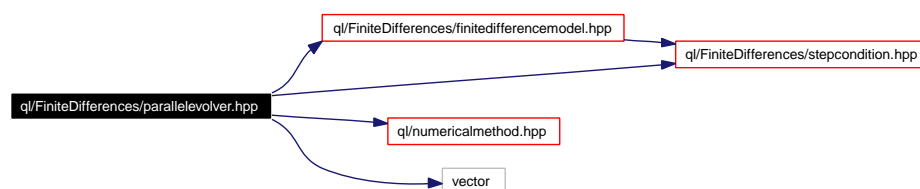
```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
```

```
#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.*

## 8.95 ql/FiniteDifferences/pde.hpp File Reference

### 8.95.1 Detailed Description

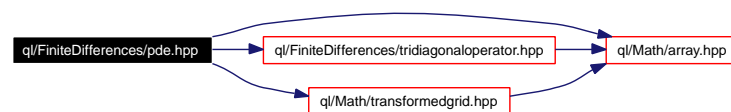
General class for one dimensional PDE's.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/Math/transformedgrid.hpp>
```

Include dependency graph for pde.hpp:



### Namespaces

- namespace **QuantLib**

## 8.96 ql/FiniteDifferences/pdebsm.hpp File Reference

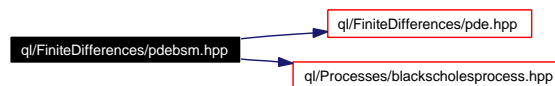
### 8.96.1 Detailed Description

Black-Scholes-Merton PDE.

```
#include <ql/FiniteDifferences/pde.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for pdebsm.hpp:



### Namespaces

- namespace **QuantLib**

## 8.97 ql/FiniteDifferences/pdeshortrate.hpp File Reference

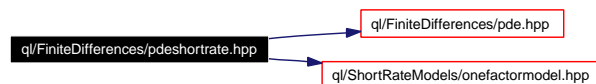
### 8.97.1 Detailed Description

adapter to short rate

```
#include <ql/FiniteDifferences/pde.hpp>
```

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for pdeshortrate.hpp:



### Namespaces

- namespace **QuantLib**

## 8.98 ql/FiniteDifferences/shoutcondition.hpp File Reference

### 8.98.1 Detailed Description

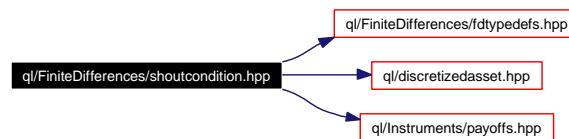
shout option exercise condition

```
#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.*

## 8.99 ql/FiniteDifferences/stepcondition.hpp File Reference

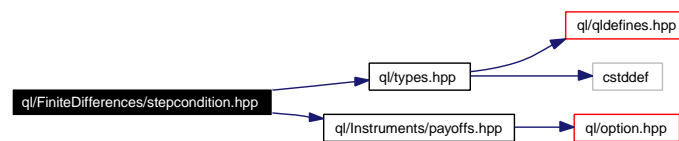
### 8.99.1 Detailed Description

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*

## 8.100 ql/FiniteDifferences/tridiagonaloperator.hpp File Reference

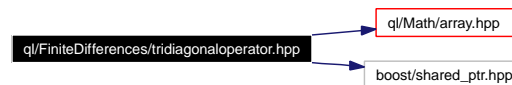
### 8.100.1 Detailed Description

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 **QuantLib::swap** (TridiagonalOperator &, TridiagonalOperator &)
- Disposable< TridiagonalOperator > **QuantLib::operator+** (const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **QuantLib::operator-** (const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **QuantLib::operator+** (const TridiagonalOperator &D1, const TridiagonalOperator &D2)
- Disposable< TridiagonalOperator > **QuantLib::operator-** (const TridiagonalOperator &D1, const TridiagonalOperator &D2)
- Disposable< TridiagonalOperator > **QuantLib::operator \*** ([Real](#) a, const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **QuantLib::operator \*** (const TridiagonalOperator &D, [Real](#) a)
- Disposable< TridiagonalOperator > **QuantLib::operator/** (const TridiagonalOperator &D, [Real](#) a)



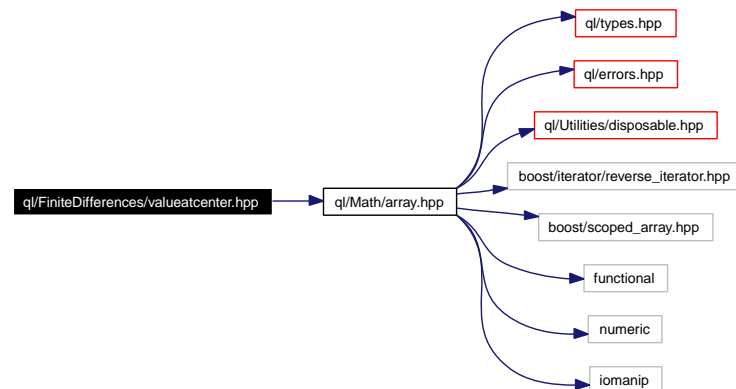
## 8.101 ql/FiniteDifferences/valueatcenter.hpp File Reference

### 8.101.1 Detailed Description

compute value, first, and second derivatives at grid center

```
#include <ql/Math/array.hpp>
```

Include dependency graph for valueatcenter.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- `template<class T> Real QuantLib::valueAtCenter` (const T &a)
- `template<class T> Real QuantLib::firstDerivativeAtCenter` (const T &a, const T &g)
- `template<class T> Real QuantLib::secondDerivativeAtCenter` (const T &a, const T &g)

## 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.*

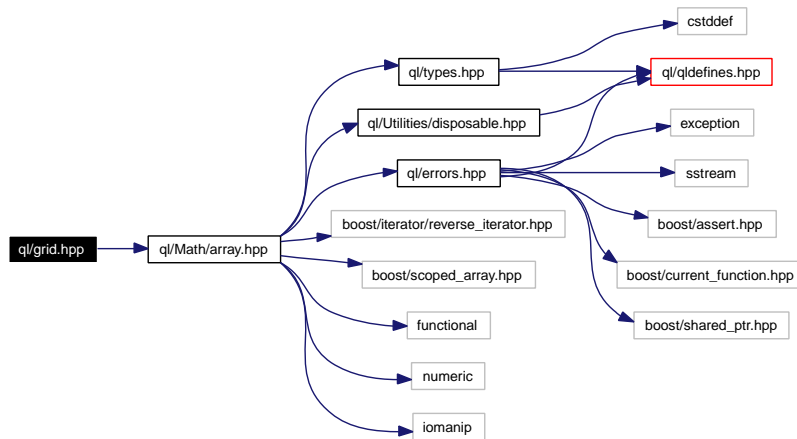
## 8.103 ql/grid.hpp File Reference

### 8.103.1 Detailed Description

Grid constructors.

```
#include <ql/Math/array.hpp>
```

Include dependency graph for grid.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- Disposable< Array > **QuantLib::CenteredGrid** (Real center, Real dx, Size steps)
- Disposable< Array > **QuantLib::BoundedGrid** (Real xMin, Real xMax, Size steps)
- Disposable< Array > **QuantLib::BoundedLogGrid** (Real xMin, Real xMax, Size steps)

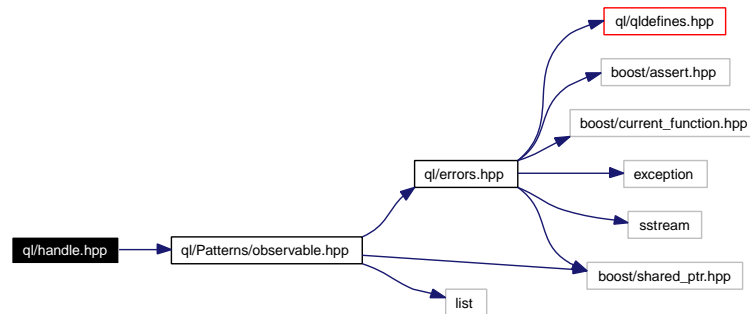
## 8.104 ql/handle.hpp File Reference

### 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/history.hpp File Reference

### 8.105.1 Detailed Description

history class

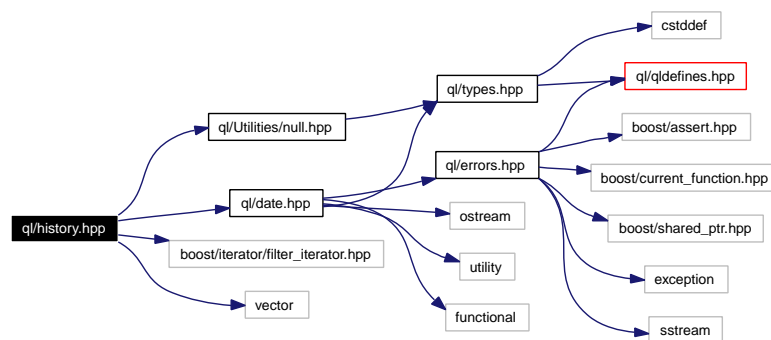
```
#include <ql/date.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

```
#include <boost/iterator/filter_iterator.hpp>
```

```
#include <vector>
```

Include dependency graph for history.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [History](#)  
*Container for historical data.*
- class [History::Entry](#)  
*single datum in history*
- class [History::const\\_iterator](#)  
*random access iterator on history entries*

## 8.106 ql/index.hpp File Reference

### 8.106.1 Detailed Description

purely virtual base class for indexes

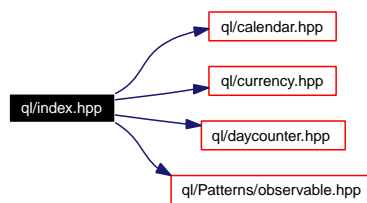
```
#include <ql/calendar.hpp>
```

```
#include <ql/currency.hpp>
```

```
#include <ql/daycounter.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for index.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Index](#)  
*purely virtual base class for indexes*

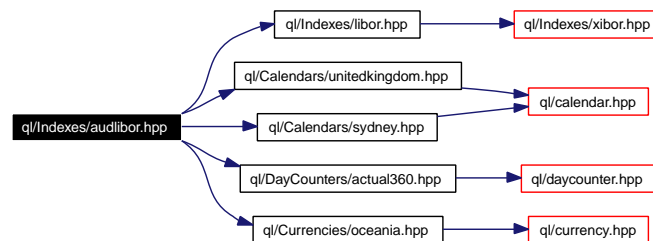
## 8.107 ql/Indexes/audlibor.hpp File Reference

### 8.107.1 Detailed Description

AUD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/sydney.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*

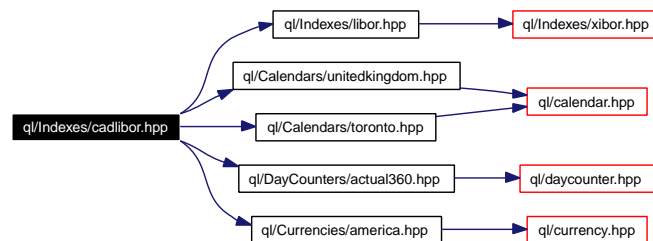
## 8.108 ql/Indexes/cadlibor.hpp File Reference

### 8.108.1 Detailed Description

CAD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/toronto.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*



## 8.109 ql/Indexes/cdor.hpp File Reference

### 8.109.1 Detailed Description

CDOR rate

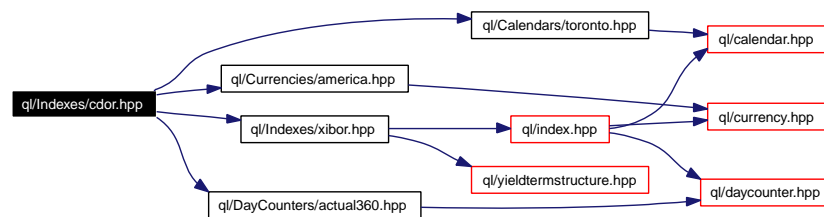
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/toronto.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*

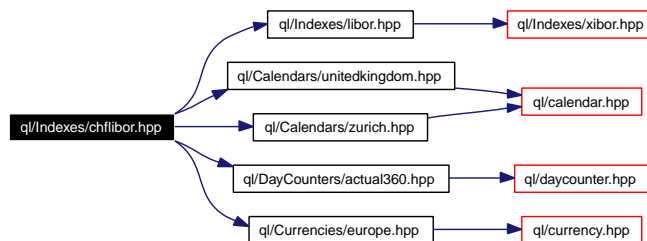
## 8.110 ql/Indexes/chflibor.hpp File Reference

### 8.110.1 Detailed Description

CHF LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/zurich.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*

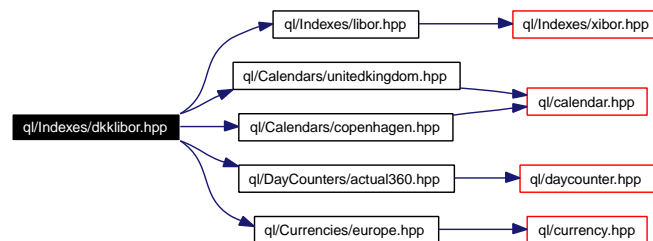
## 8.111 ql/Indexes/dkklibor.hpp File Reference

### 8.111.1 Detailed Description

DKK LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/copenhagen.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*

## 8.112 ql/Indexes/euribor.hpp File Reference

### 8.112.1 Detailed Description

Euribor index

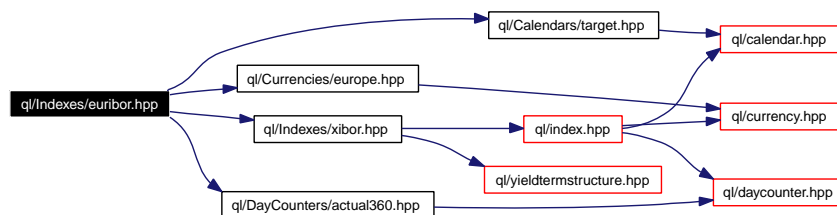
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for euribor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Euribor**  
*Euribor index*

## 8.113 ql/Indexes/eurlibor.hpp File Reference

### 8.113.1 Detailed Description

EUR LIBOR rate

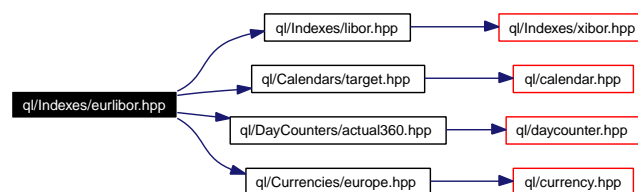
```
#include <ql/Indexes/libor.hpp>
```

```
#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*

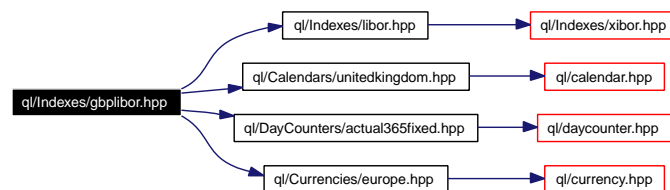
## 8.114 ql/Indexes/gbplibor.hpp File Reference

### 8.114.1 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*

## 8.115 ql/Indexes/indexmanager.hpp File Reference

### 8.115.1 Detailed Description

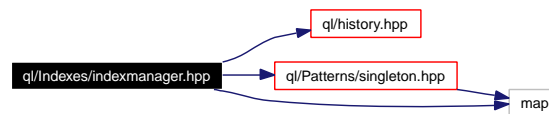
global repository for past index fixings

```
#include <ql/history.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <map>
```

Include dependency graph for indexmanager.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **IndexManager**  
*global repository for past index fixings*

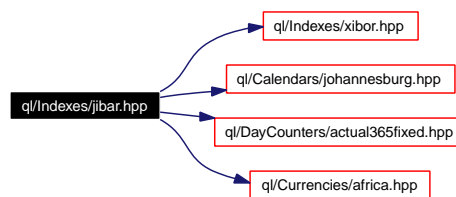
## 8.116 ql/Indexes/jibar.hpp File Reference

### 8.116.1 Detailed Description

JIBAR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/johannesburg.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*



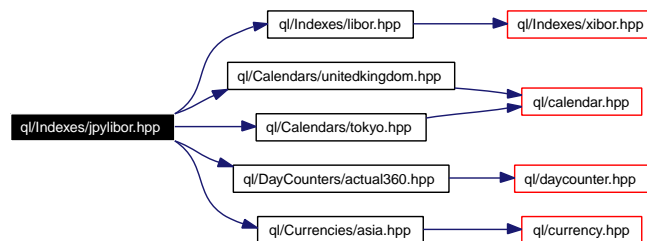
## 8.117 ql/Indexes/jpylibor.hpp File Reference

### 8.117.1 Detailed Description

JPY LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/tokyo.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*

## 8.118 ql/Indexes/libor.hpp File Reference

### 8.118.1 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*

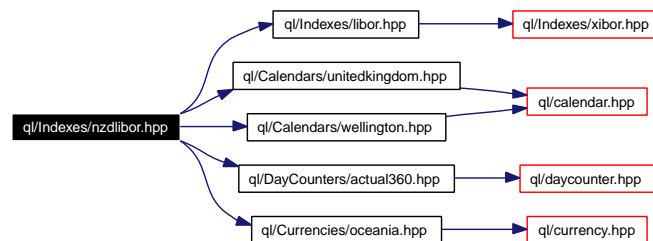
## 8.119 ql/Indexes/nzdlbor.hpp File Reference

### 8.119.1 Detailed Description

NZD LIBOR rate

```
#include <ql/Indexes/libor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/Calendars/wellington.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/oceania.hpp>
```

Include dependency graph for nzdlbor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **NZDLibor**  
*NZD LIBOR rate*

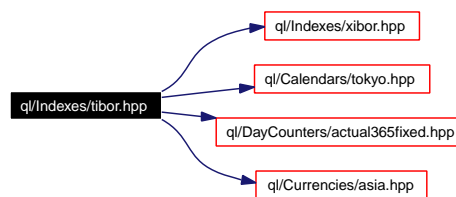
## 8.120 ql/Indexes/tibor.hpp File Reference

### 8.120.1 Detailed Description

JPY TIBOR rate

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/tokyo.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*

## 8.121 ql/Indexes/trlibor.hpp File Reference

### 8.121.1 Detailed Description

TRY LIBOR rate

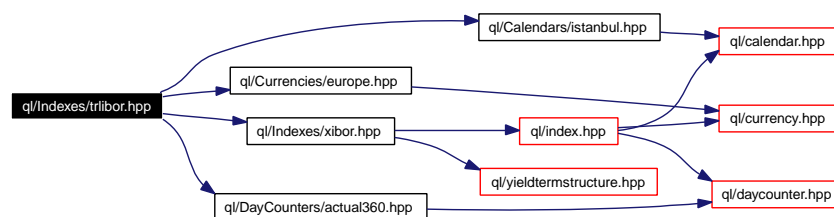
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/istanbul.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*

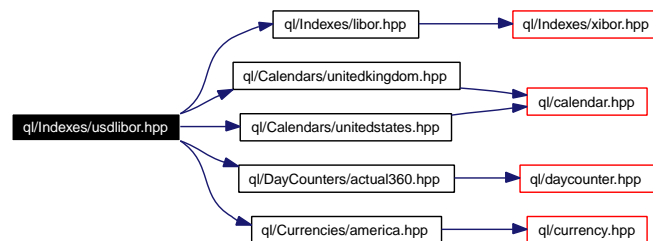
## 8.122 ql/Indexes/usdlibor.hpp File Reference

### 8.122.1 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*

## 8.123 ql/Indexes/xibor.hpp File Reference

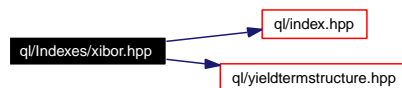
### 8.123.1 Detailed Description

base class for LIBOR-like indexes

```
#include <ql/index.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

Include dependency graph for xibor.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Xibor](#)  
*base class for LIBOR-like indexes*

## 8.124 ql/Indexes/zibor.hpp File Reference

### 8.124.1 Detailed Description

CHF ZIBOR rate

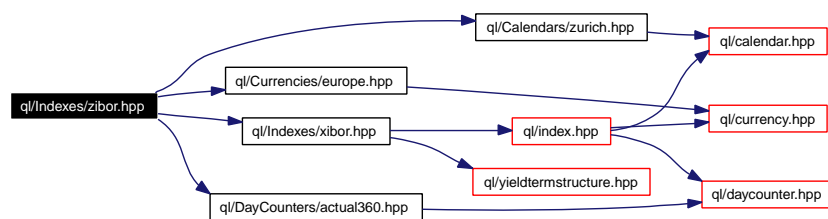
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/zurich.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



## 8.125 ql/instrument.hpp File Reference

### 8.125.1 Detailed Description

Abstract instrument class.

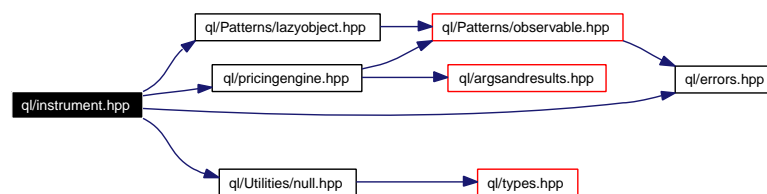
```
#include <ql/Patterns/lazyobject.hpp>
```

```
#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*

## 8.126 ql/Instruments/asianoption.hpp File Reference

### 8.126.1 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.*

## 8.127 ql/Instruments/barrieroption.hpp File Reference

### 8.127.1 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*

## 8.128 ql/Instruments/basketoption.hpp File Reference

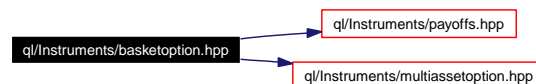
### 8.128.1 Detailed Description

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*

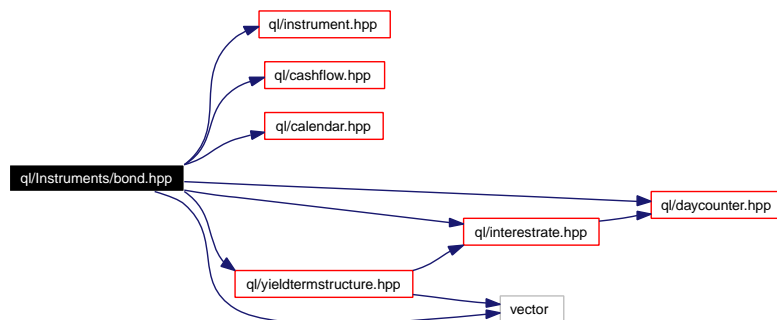
## 8.129 ql/Instruments/bond.hpp File Reference

### 8.129.1 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.*

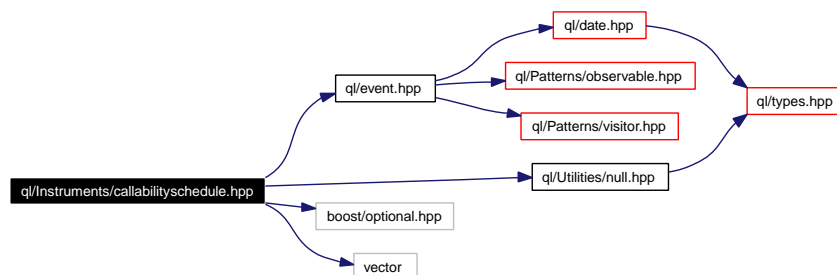
## 8.130 ql/Instruments/callabilityschedule.hpp File Reference

### 8.130.1 Detailed Description

Schedule of put/call dates.

```
#include <ql/event.hpp>
#include <ql/Utilities/null.hpp>
#include <boost/optional.hpp>
#include <vector>
```

Include dependency graph for callabilityschedule.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `std::vector< Callability >` **QuantLib::CallabilitySchedule**

## 8.131 ql/Instruments/capfloor.hpp File Reference

### 8.131.1 Detailed Description

Cap and Floor class.

```
#include <ql/numericalmethod.hpp>
```

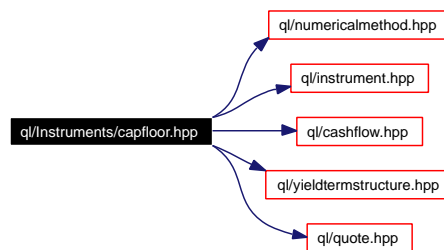
```
#include <ql/instrument.hpp>
```

```
#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*

## 8.132 ql/Instruments/cliquestoption.hpp File Reference

### 8.132.1 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.*



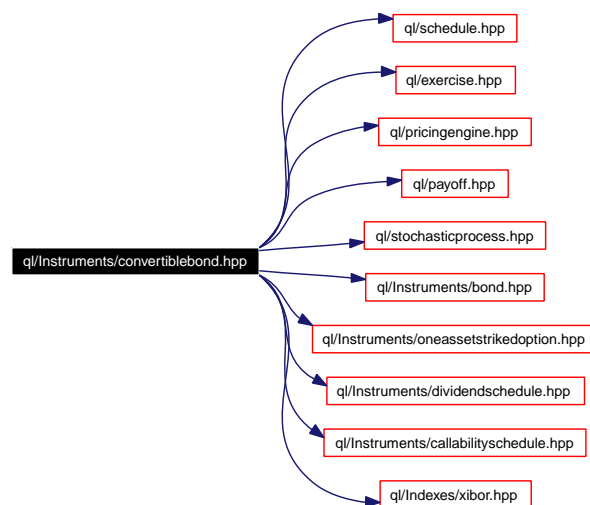
## 8.133 ql/Instruments/convertiblebond.hpp File Reference

### 8.133.1 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 **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*

## 8.134 ql/Instruments/dividendschedule.hpp File Reference

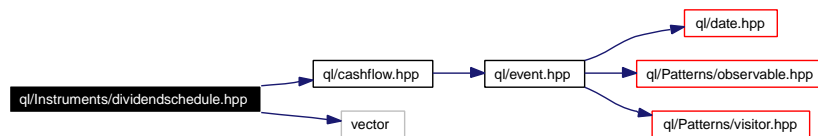
### 8.134.1 Detailed Description

Schedule of dividend dates.

```
#include <ql/cashflow.hpp>
```

```
#include <vector>
```

Include dependency graph for dividendschedule.hpp:



### Namespaces

- namespace **QuantLib**

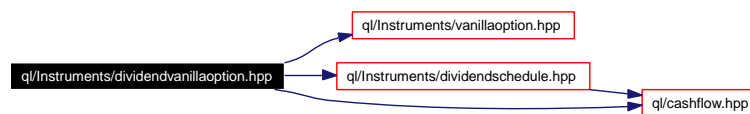
## 8.135 ql/Instruments/dividendvanillaoption.hpp File Reference

### 8.135.1 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.*

## 8.136 ql/Instruments/europeanoption.hpp File Reference

### 8.136.1 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.*

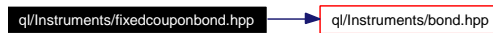
## 8.137 ql/Instruments/fixedcouponbond.hpp File Reference

### 8.137.1 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*

## 8.138 ql/Instruments/floatingratebond.hpp File Reference

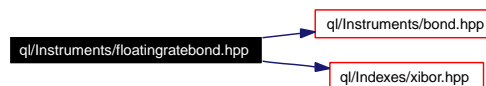
### 8.138.1 Detailed Description

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*

## 8.139 ql/Instruments/forwardvanillaoption.hpp File Reference

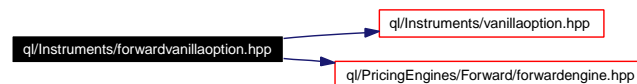
### 8.139.1 Detailed Description

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.*



## 8.140 ql/Instruments/multiassetoption.hpp File Reference

### 8.140.1 Detailed Description

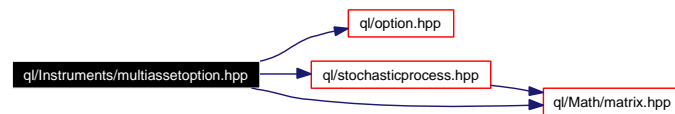
Option on multiple assets.

```
#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*

## 8.141 ql/Instruments/oneassetoption.hpp File Reference

### 8.141.1 Detailed Description

Option on a single asset.

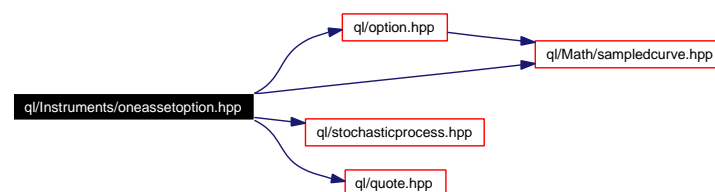
```
#include <ql/option.hpp>
```

```
#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*

## 8.142 ql/Instruments/oneassetstrikedoption.hpp File Reference

### 8.142.1 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.*

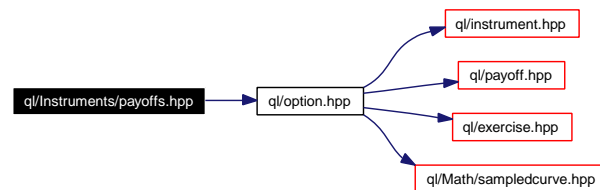
## 8.143 ql/Instruments/payoffs.hpp File Reference

### 8.143.1 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 [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.*

## 8.144 ql/Instruments/quantoforwardvanillaoption.hpp File Reference

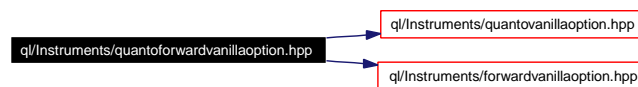
### 8.144.1 Detailed Description

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.*

## 8.145 ql/Instruments/quantovanillaoption.hpp File Reference

### 8.145.1 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*

## 8.146 ql/Instruments/simpleswap.hpp File Reference

### 8.146.1 Detailed Description

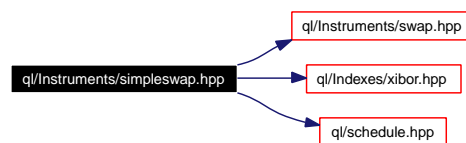
Simple fixed-rate vs Libor swap.

```
#include <ql/Instruments/swap.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/schedule.hpp>
```

Include dependency graph for simpleswap.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*

### Typedefs

- typedef VanillaSwap [QuantLib::SimpleSwap](#)

## 8.147 ql/Instruments/stock.hpp File Reference

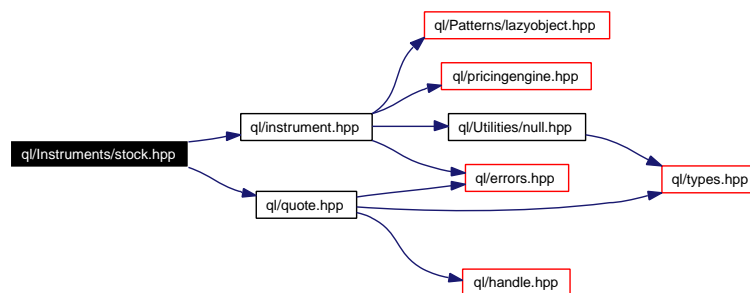
### 8.147.1 Detailed Description

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.*



## 8.148 ql/Instruments/swap.hpp File Reference

### 8.148.1 Detailed Description

Interest rate swap.

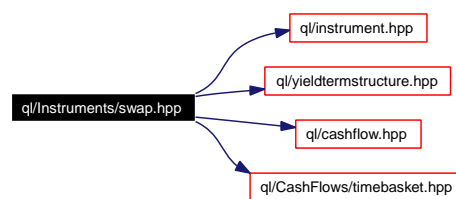
```
#include <ql/instrument.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/cashflow.hpp>
```

```
#include <ql/CashFlows/timebasket.hpp>
```

Include dependency graph for swap.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Swap](#)  
*Interest rate swap.*

## 8.149 ql/Instruments/swaption.hpp File Reference

### 8.149.1 Detailed Description

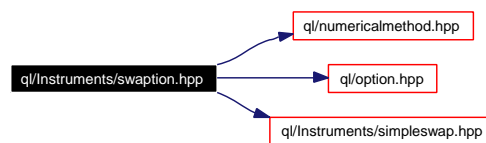
Swaption class.

```
#include <ql/numericalmethod.hpp>
```

```
#include <ql/option.hpp>
```

```
#include <ql/Instruments/simpleswap.hpp>
```

Include dependency graph for swaption.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **Swaption**  
*Swaption class*
- class **Swaption::arguments**  
*Arguments for swaption calculation*
- class **Swaption::results**  
*Results from swaption calculation*

## 8.150 ql/Instruments/vanillaoption.hpp File Reference

### 8.150.1 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.*

## 8.151 ql/Instruments/zerocouponbond.hpp File Reference

### 8.151.1 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*

## 8.152 ql/interestrate.hpp File Reference

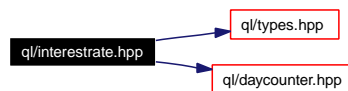
### 8.152.1 Detailed Description

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** { [QuantLib::Simple](#) = 0, [QuantLib::Compounded](#) = 1, [QuantLib::Continuous](#) = 2, [QuantLib::SimpleThenCompounded](#) }  
*Interest rate compounding rule.*

## 8.153 ql/Lattices/binomialtree.hpp File Reference

### 8.153.1 Detailed Description

Binomial tree class.

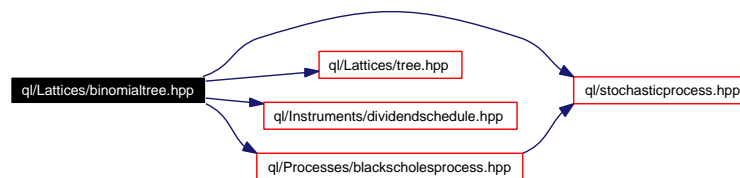
```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/Lattices/tree.hpp>
```

```
#include <ql/Instruments/dividendschedule.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for binomialtree.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.*

## 8.154 ql/Lattices/bsmlattice.hpp File Reference

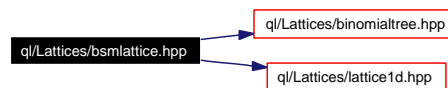
### 8.154.1 Detailed Description

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.*



## 8.155 ql/Lattices/lattice.hpp File Reference

### 8.155.1 Detailed Description

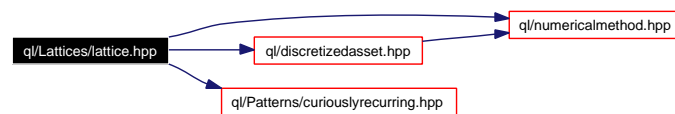
Lattice method class.

```
#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.*

## 8.156 ql/Lattices/lattice1d.hpp File Reference

### 8.156.1 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.*

## 8.157 ql/Lattices/lattice2d.hpp File Reference

### 8.157.1 Detailed Description

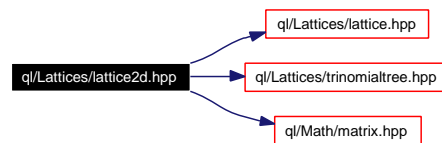
Two-dimensional lattice class.

```
#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.*

## 8.158 ql/Lattices/tflattice.hpp File Reference

### 8.158.1 Detailed Description

Binomial Tsiveriotis-Fernandes tree model.

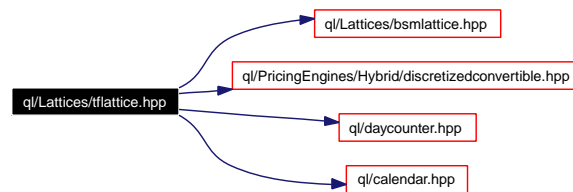
```
#include <ql/Lattices/bsmlattice.hpp>
```

```
#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.*

## 8.159 ql/Lattices/tree.hpp File Reference

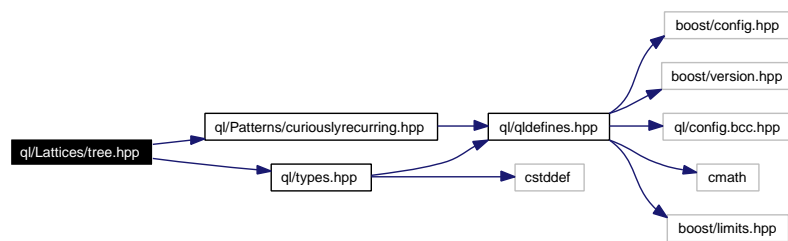
### 8.159.1 Detailed Description

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*

## 8.160 ql/Lattices/trinomialtree.hpp File Reference

### 8.160.1 Detailed Description

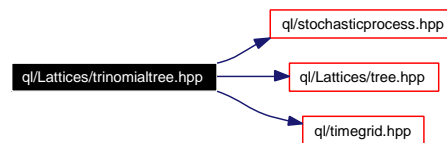
Trinomial tree class.

```
#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.*

## 8.161 ql/Math/array.hpp File Reference

### 8.161.1 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>
```

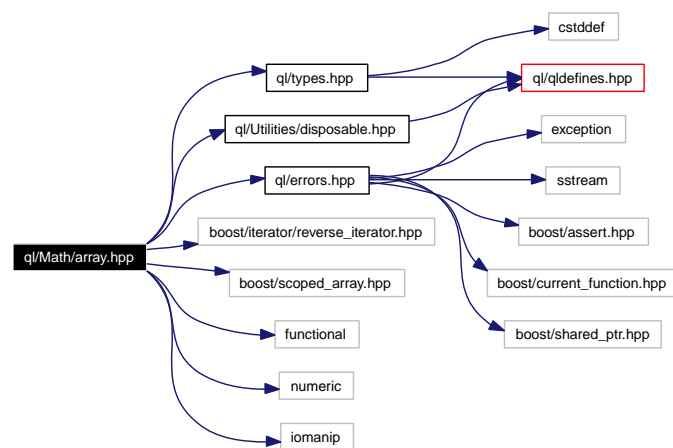
```
#include <boost/scoped_array.hpp>
```

```
#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.*

## 8.162 ql/Math/backwardflatinterpolation.hpp File Reference

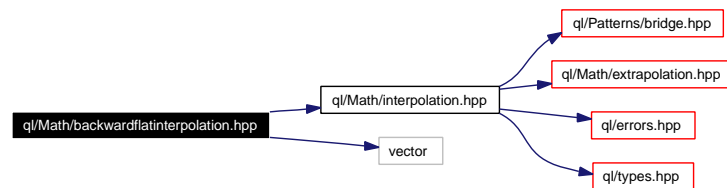
### 8.162.1 Detailed Description

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.*



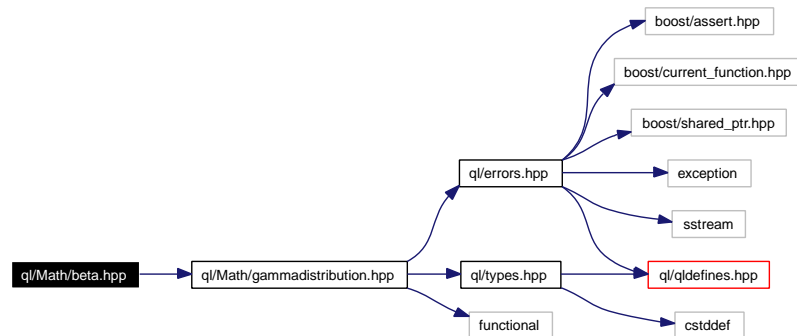
## 8.163 ql/Math/beta.hpp File Reference

### 8.163.1 Detailed Description

Beta and beta incomplete functions.

```
#include <ql/Math/gammadistribution.hpp>
```

Include dependency graph for beta.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- **Real** **QuantLib::betaFunction** (**Real** z, **Real** w)
- **Real** **QuantLib::betaContinuedFraction** (**Real** a, **Real** b, **Real** x, **Real** accuracy=1e-16, Integer maxIteration=100)
- **Real** **QuantLib::incompleteBetaFunction** (**Real** a, **Real** b, **Real** x, **Real** accuracy=1e-16, Integer maxIteration=100)

*Incomplete Beta function.*

## 8.164 ql/Math/bicubicsplineinterpolation.hpp File Reference

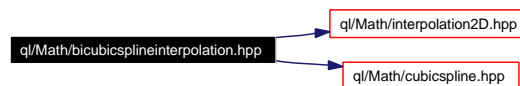
### 8.164.1 Detailed Description

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*

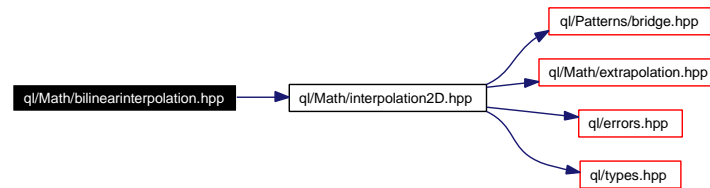
## 8.165 ql/Math/bilinearinterpolation.hpp File Reference

### 8.165.1 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*

## 8.166 ql/Math/binomialdistribution.hpp File Reference

### 8.166.1 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 QuantLib::binomialCoefficientLn](#) (BigNatural n, BigNatural k)
- [Real QuantLib::binomialCoefficient](#) (BigNatural n, BigNatural k)
- [Real QuantLib::PeizerPrattMethod2Inversion](#) (Real z, BigNatural n)

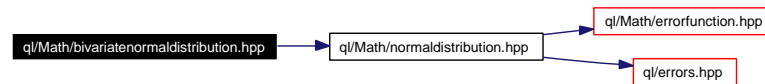
## 8.167 ql/Math/bivariatenormaldistribution.hpp File Reference

### 8.167.1 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](#) [QuantLib::BivariateCumulativeNormalDistribution](#)  
*default bivariate implementation*

## 8.168 ql/Math/chisquaredistribution.hpp File Reference

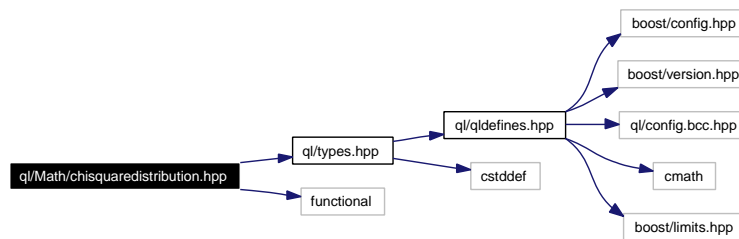
### 8.168.1 Detailed Description

Chi-square (central and non-central) distributions.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for chisquaredistribution.hpp:



### Namespaces

- namespace **QuantLib**

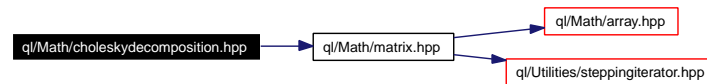
## 8.169 ql/Math/choleskydecomposition.hpp File Reference

### 8.169.1 Detailed Description

Cholesky decomposition.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for choleskydecomposition.hpp:



### Namespaces

- namespace **QuantLib**

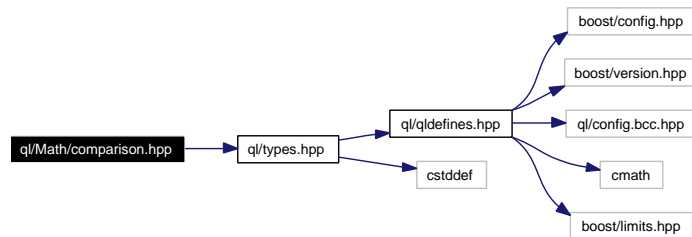
## 8.170 ql/Math/comparison.hpp File Reference

### 8.170.1 Detailed Description

floating-point comparisons

```
#include <ql/types.hpp>
```

Include dependency graph for comparison.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- bool [QuantLib::close](#) ([Real](#) x, [Real](#) y)
- bool **QuantLib::close** ([Real](#) x, [Real](#) y, [Size](#) n)
- bool [QuantLib::close\\_enough](#) ([Real](#) x, [Real](#) y)
- bool **QuantLib::close\_enough** ([Real](#) x, [Real](#) y, [Size](#) n)



## 8.171 ql/Math/convergencestatistics.hpp File Reference

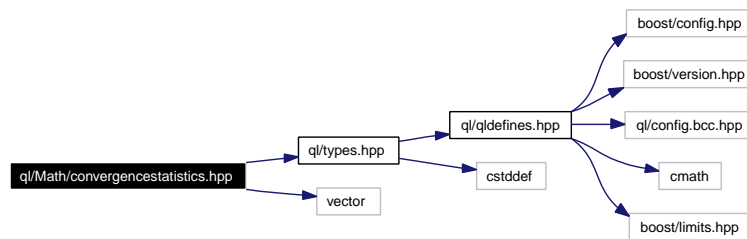
### 8.171.1 Detailed Description

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*

## 8.172 ql/Math/cubicspline.hpp File Reference

### 8.172.1 Detailed Description

cubic spline interpolation between discrete points

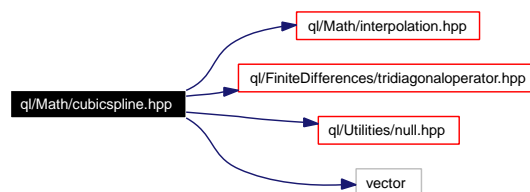
```
#include <ql/Math/interpolation.hpp>
```

```
#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*

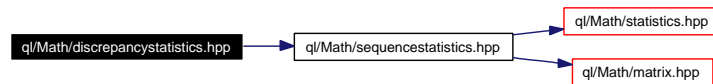
## 8.173 ql/Math/discrepancystatistics.hpp File Reference

### 8.173.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

```
#include <ql/Math/sequencestatistics.hpp>
```

Include dependency graph for `discrepancystatistics.hpp`:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DiscrepancyStatistics](#)  
*Statistic tool for sequences with discrepancy calculation.*

## 8.174 ql/Math/errorfunction.hpp File Reference

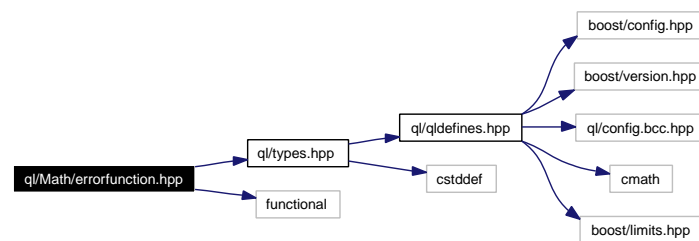
### 8.174.1 Detailed Description

Error function.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for errorfunction.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [ErrorFunction](#)

*Error function*

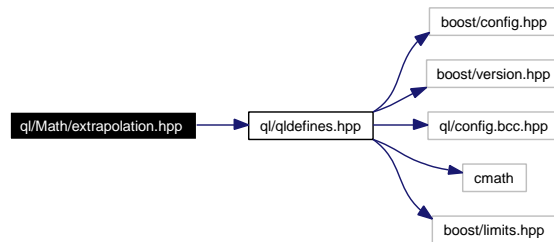
## 8.175 ql/Math/extrapolation.hpp File Reference

### 8.175.1 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*

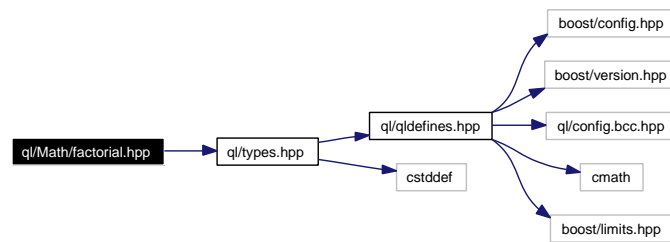
## 8.176 ql/Math/factorial.hpp File Reference

### 8.176.1 Detailed Description

Factorial numbers calculator.

```
#include <ql/types.hpp>
```

Include dependency graph for factorial.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [Factorial](#)  
*Factorial numbers calculator*

## 8.177 ql/Math/forwardflatinterpolation.hpp File Reference

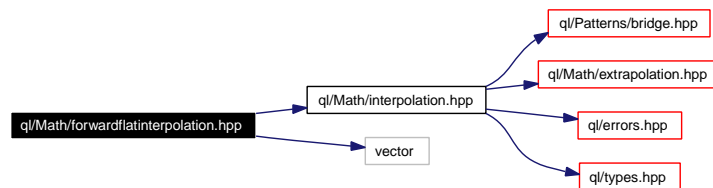
### 8.177.1 Detailed Description

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.*

## 8.178 ql/Math/functional.hpp File Reference

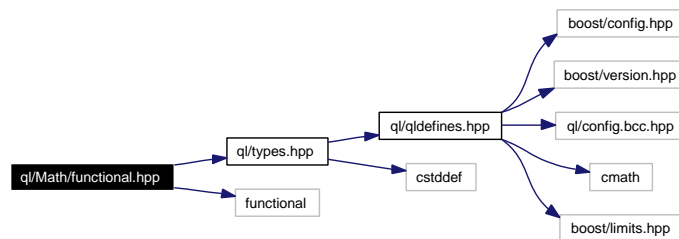
### 8.178.1 Detailed Description

functionals and combinators not included in the STL

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for functional.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- `template<class F, class R> clipped_function< F, R > QuantLib::clip (const F &f, const R &r)`
- `template<class F, class G> composed_function< F, G > QuantLib::compose (const F &f, const G &g)`



## 8.179 ql/Math/gammadistribution.hpp File Reference

### 8.179.1 Detailed Description

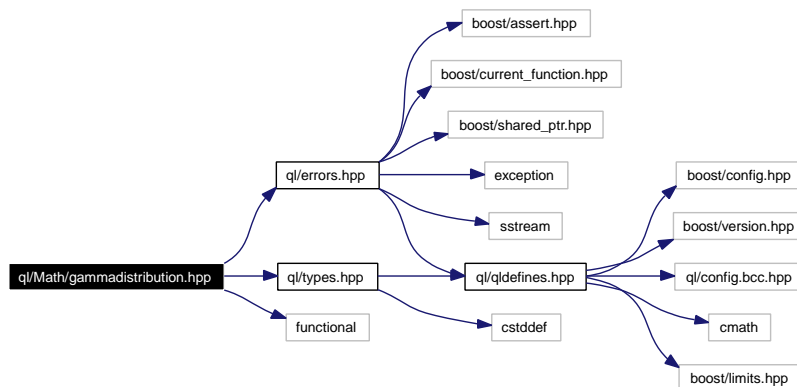
Gamma distribution.

```
#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.*

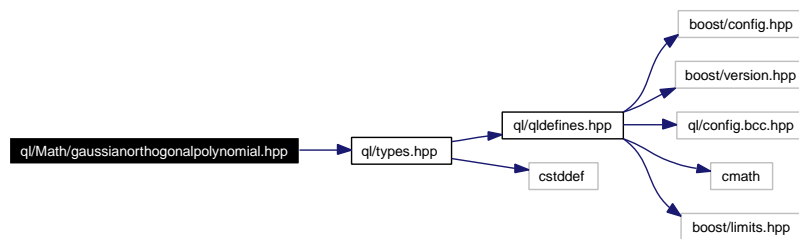
## 8.180 `ql/Math/gaussianorthogonalpolynomial.hpp` File Reference

### 8.180.1 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.*

## 8.181 ql/Math/gaussianquadratures.hpp File Reference

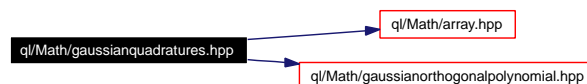
### 8.181.1 Detailed Description

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*

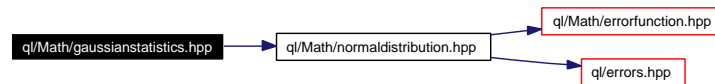
## 8.182 ql/Math/gaussianstatistics.hpp File Reference

### 8.182.1 Detailed Description

statistics tool for gaussian-assumption risk measures

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for gaussianstatistics.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GaussianStatistics](#)  
*Statistics tool for gaussian-assumption risk measures.*
- class [StatsHolder](#)  
*Helper class for precomputed distributions.*

## 8.183 ql/Math/generalstatistics.hpp File Reference

### 8.183.1 Detailed Description

statistics tool

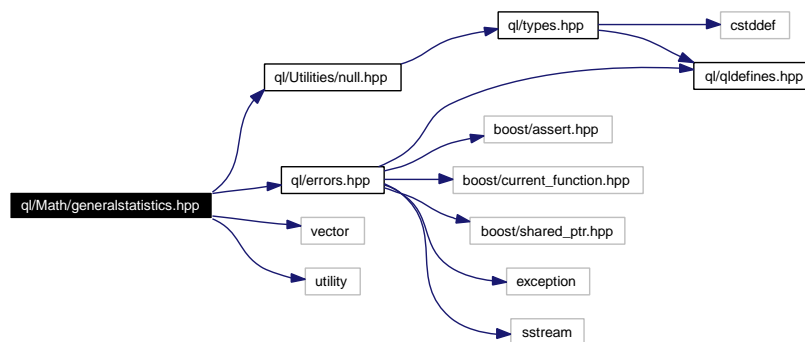
```
#include <ql/Utilities/null.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <vector>
```

```
#include <utility>
```

Include dependency graph for generalstatistics.hpp:



### Namespaces

- namespace `QuantLib`

### Classes

- class `GeneralStatistics`  
*Statistics tool.*

## 8.184 ql/Math/incompletegamma.hpp File Reference

### 8.184.1 Detailed Description

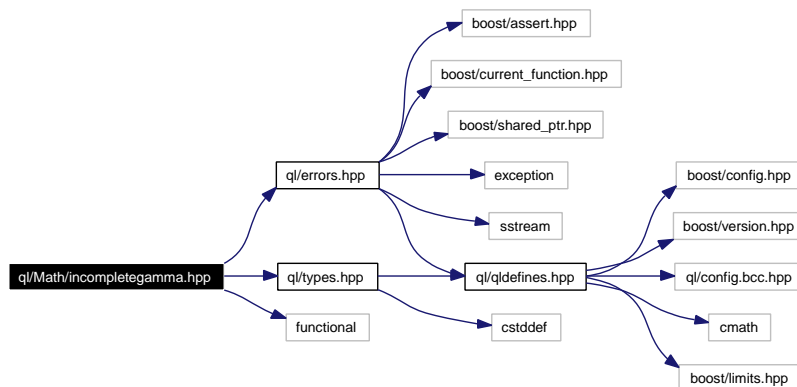
Incomplete Gamma function.

```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for incompletegamma.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- **Real** **QuantLib::incompleteGammaFunction** (**Real** a, **Real** x, **Real** accuracy=1.0e-13, Integer maxIteration=100)  
*Incomplete Gamma function.*
- **Real** **QuantLib::incompleteGammaFunctionSeriesRepr** (**Real** a, **Real** x, **Real** accuracy=1.0e-13, Integer maxIteration=100)
- **Real** **QuantLib::incompleteGammaFunctionContinuedFractionRepr** (**Real** a, **Real** x, **Real** accuracy=1.0e-13, Integer maxIteration=100)

## 8.185 ql/Math/incrementalstatistics.hpp File Reference

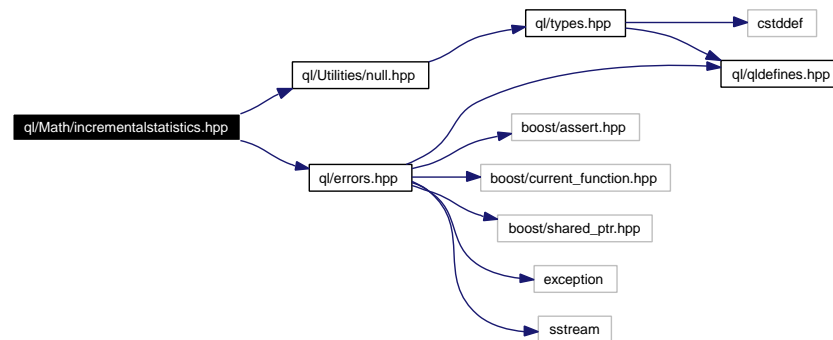
### 8.185.1 Detailed Description

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.*



## 8.186 ql/Math/interpolation.hpp File Reference

### 8.186.1 Detailed Description

base class for 1-D interpolations

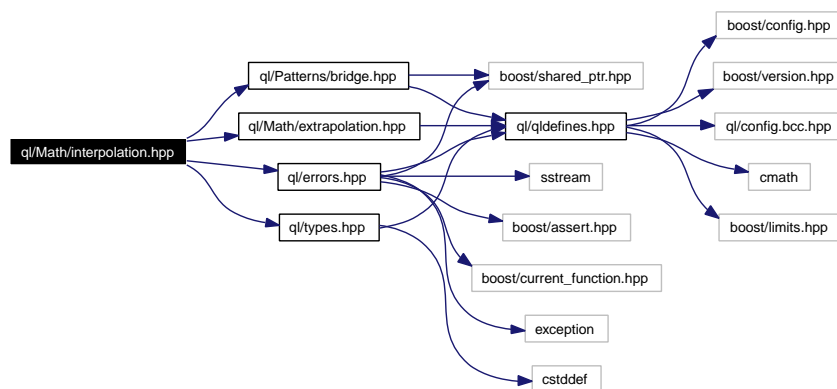
```
#include <ql/Patterns/bridge.hpp>
```

```
#include <ql/Math/extrapolation.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*

## 8.187 ql/Math/interpolation2D.hpp File Reference

### 8.187.1 Detailed Description

abstract base classes for 2-D interpolations

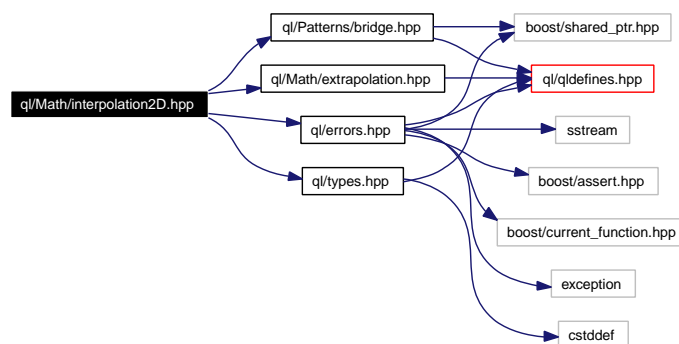
```
#include <ql/Patterns/bridge.hpp>
```

```
#include <ql/Math/extrapolation.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

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*

## 8.188 ql/Math/kronrodintegral.hpp File Reference

### 8.188.1 Detailed Description

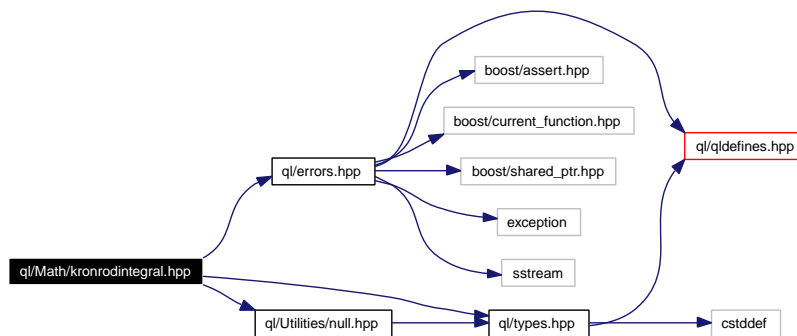
Integral of a 1-dimensional function using the Gauss-Kronrod method.

```
#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.*

## 8.189 ql/Math/lexicographicalview.hpp File Reference

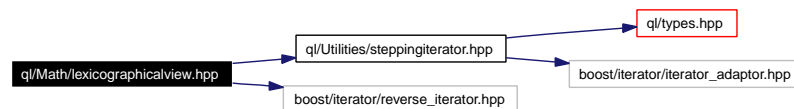
### 8.189.1 Detailed Description

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.*

## 8.190 ql/Math/linearinterpolation.hpp File Reference

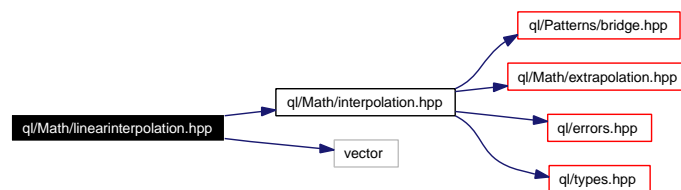
### 8.190.1 Detailed Description

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.*

## 8.191 ql/Math/loglinearinterpolation.hpp File Reference

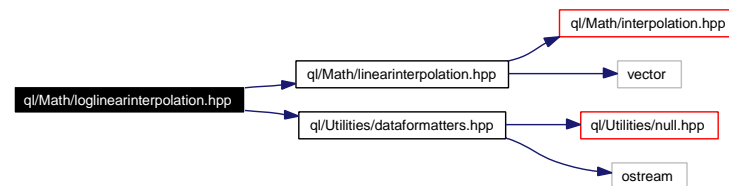
### 8.191.1 Detailed Description

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*

## 8.192 ql/Math/matrix.hpp File Reference

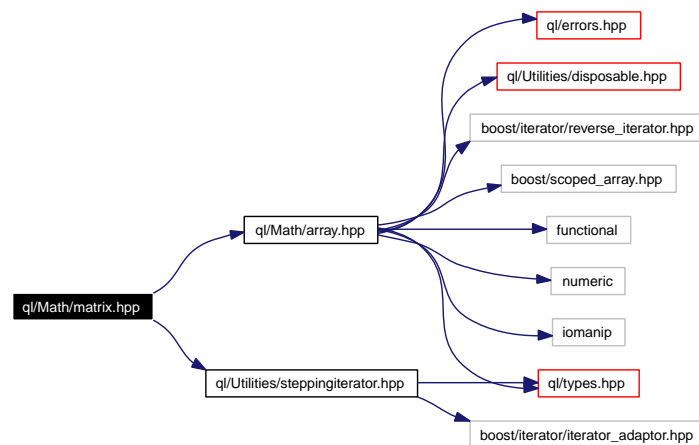
### 8.192.1 Detailed Description

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.*

## 8.193 ql/Math/multicubicspline.hpp File Reference

### 8.193.1 Detailed Description

N-dimensional cubic spline interpolation between discrete points.

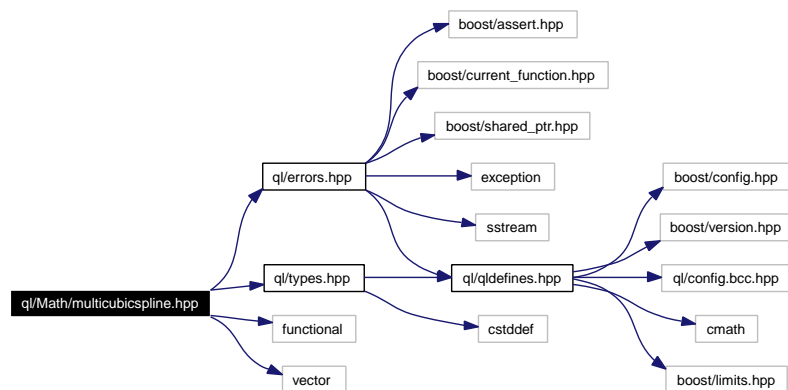
```
#include <ql/errors.hpp>
```

```
#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 > >` **QuantLib::detail::SplineGrid**
- typedef `DataTable< Real >` **QuantLib::detail::base\_data\_table**
- typedef `Data< std::vector< Real >, EmptyArg >` **QuantLib::detail::base\_data**
- typedef `Point< Real, EmptyArg >` **QuantLib::detail::base\_arg\_type**
- typedef `Point< Real, EmptyRes >` **QuantLib::detail::base\_return\_type**
- typedef `Point< Size, EmptyDim >` **QuantLib::detail::base\_dimensions**
- typedef `Point< base_data_table, EmptyRes >` **QuantLib::detail::base\_output\_data**
- typedef `base_cubic_spline` **QuantLib::detail::cubic\_spline\_01**
- typedef `n_cubic_spline< cubic_spline_01 >` **QuantLib::detail::cubic\_spline\_02**
- typedef `n_cubic_spline< cubic_spline_02 >` **QuantLib::detail::cubic\_spline\_03**
- typedef `n_cubic_spline< cubic_spline_03 >` **QuantLib::detail::cubic\_spline\_04**



- typedef n\_cubic\_spline< cubic\_spline\_04 > QuantLib::detail::cubic\_spline\_05
- typedef n\_cubic\_spline< cubic\_spline\_05 > QuantLib::detail::cubic\_spline\_06
- typedef n\_cubic\_spline< cubic\_spline\_06 > QuantLib::detail::cubic\_spline\_07
- typedef n\_cubic\_spline< cubic\_spline\_07 > QuantLib::detail::cubic\_spline\_08
- typedef n\_cubic\_spline< cubic\_spline\_08 > QuantLib::detail::cubic\_spline\_09
- typedef n\_cubic\_spline< cubic\_spline\_09 > QuantLib::detail::cubic\_spline\_10
- typedef n\_cubic\_spline< cubic\_spline\_10 > QuantLib::detail::cubic\_spline\_11
- typedef n\_cubic\_spline< cubic\_spline\_11 > QuantLib::detail::cubic\_spline\_12
- typedef n\_cubic\_spline< cubic\_spline\_12 > QuantLib::detail::cubic\_spline\_13
- typedef n\_cubic\_spline< cubic\_spline\_13 > QuantLib::detail::cubic\_spline\_14
- typedef n\_cubic\_spline< cubic\_spline\_14 > QuantLib::detail::cubic\_spline\_15
- typedef base\_cubic\_splint QuantLib::detail::cubic\_splint\_01
- typedef n\_cubic\_splint< cubic\_splint\_01 > QuantLib::detail::cubic\_splint\_02
- typedef n\_cubic\_splint< cubic\_splint\_02 > QuantLib::detail::cubic\_splint\_03
- typedef n\_cubic\_splint< cubic\_splint\_03 > QuantLib::detail::cubic\_splint\_04
- typedef n\_cubic\_splint< cubic\_splint\_04 > QuantLib::detail::cubic\_splint\_05
- typedef n\_cubic\_splint< cubic\_splint\_05 > QuantLib::detail::cubic\_splint\_06
- typedef n\_cubic\_splint< cubic\_splint\_06 > QuantLib::detail::cubic\_splint\_07
- typedef n\_cubic\_splint< cubic\_splint\_07 > QuantLib::detail::cubic\_splint\_08
- typedef n\_cubic\_splint< cubic\_splint\_08 > QuantLib::detail::cubic\_splint\_09
- typedef n\_cubic\_splint< cubic\_splint\_09 > QuantLib::detail::cubic\_splint\_10
- typedef n\_cubic\_splint< cubic\_splint\_10 > QuantLib::detail::cubic\_splint\_11
- typedef n\_cubic\_splint< cubic\_splint\_11 > QuantLib::detail::cubic\_splint\_12
- typedef n\_cubic\_splint< cubic\_splint\_12 > QuantLib::detail::cubic\_splint\_13
- typedef n\_cubic\_splint< cubic\_splint\_13 > QuantLib::detail::cubic\_splint\_14
- typedef n\_cubic\_splint< cubic\_splint\_14 > QuantLib::detail::cubic\_splint\_15
- typedef detail::SplineGrid QuantLib::SplineGrid

## 8.194 ql/Math/normaldistribution.hpp File Reference

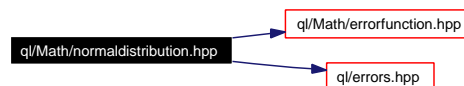
### 8.194.1 Detailed Description

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 **QuantLib::GaussianDistribution**
- typedef InverseCumulativeNormal **QuantLib::InvCumulativeNormalDistribution**

## 8.195 ql/Math/poissondistribution.hpp File Reference

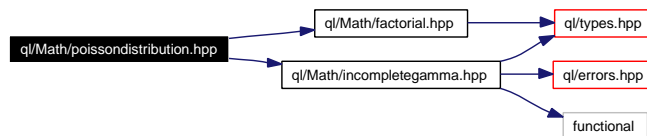
### 8.195.1 Detailed Description

Poisson distribution.

```
#include <ql/Math/factorial.hpp>
```

```
#include <ql/Math/incompletingamma.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.*

## 8.196 ql/Math/primenumbers.hpp File Reference

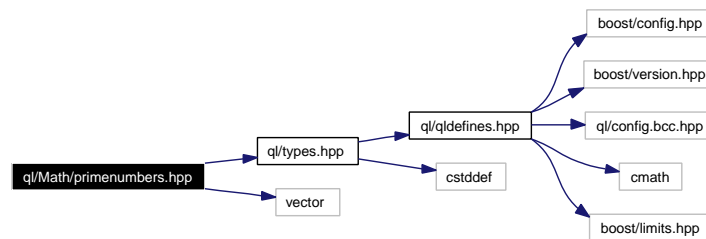
### 8.196.1 Detailed Description

Prime numbers calculator.

```
#include <ql/types.hpp>
```

```
#include <vector>
```

Include dependency graph for primenumbers.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [PrimeNumbers](#)  
*Prime numbers calculator.*

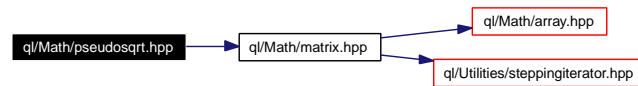
## 8.197 ql/Math/pseudosqrt.hpp File Reference

### 8.197.1 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*

## 8.198 ql/Math/riskstatistics.hpp File Reference

### 8.198.1 Detailed Description

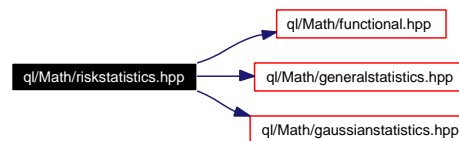
empirical-distribution risk measures

```
#include <ql/Math/functional.hpp>
```

```
#include <ql/Math/generalstatistics.hpp>
```

```
#include <ql/Math/gaussianstatistics.hpp>
```

Include dependency graph for riskstatistics.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [GenericRiskStatistics](#)  
*empirical-distribution risk measures*

### Typedefs

- typedef `GaussianStatistics< GenericRiskStatistics< GeneralStatistics > >` [QuantLib::RiskStatistics](#)  
*default risk measures tool*

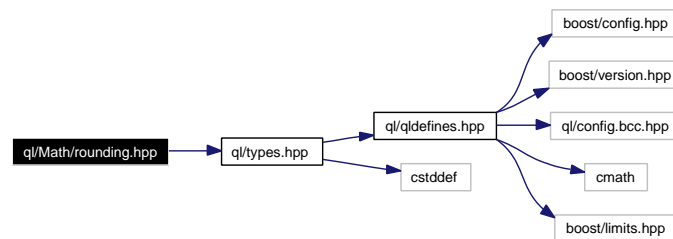
## 8.199 ql/Math/rounding.hpp File Reference

### 8.199.1 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.*

## 8.200 ql/Math/sampledcurve.hpp File Reference

### 8.200.1 Detailed Description

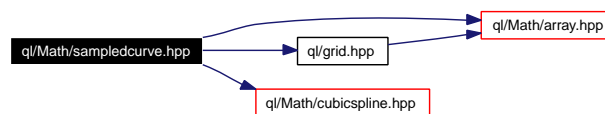
a class that contains a sampled curve

```
#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` **QuantLib::SampledCurveSet**

### Functions

- void **QuantLib::swap** (`SampledCurve &`, `SampledCurve &`)
- `std::ostream &` **QuantLib::operator<<** (`std::ostream &out`, `const SampledCurve &a`)



## 8.201 ql/Math/segmentintegral.hpp File Reference

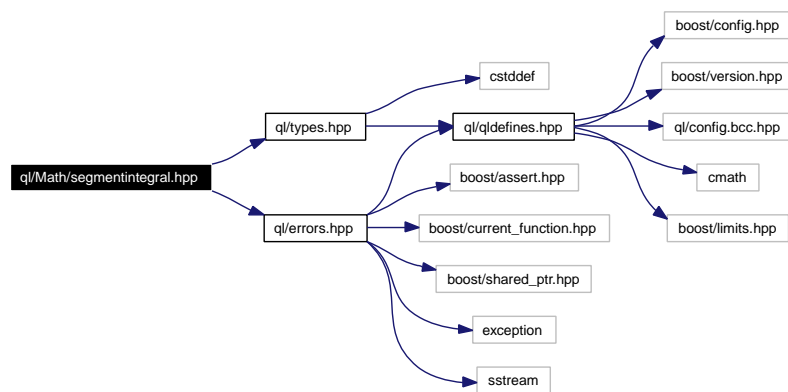
### 8.201.1 Detailed Description

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.*

## 8.202 ql/Math/sequencestatistics.hpp File Reference

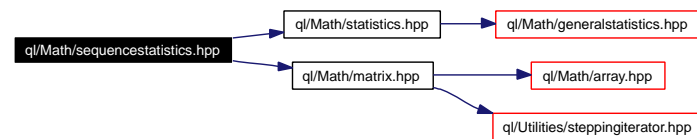
### 8.202.1 Detailed Description

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 [SequenceStatistics](#)  
*Statistics analysis of N-dimensional (sequence) data.*

### Defines

- `#define DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID(METHOD)`
- `#define DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)`

### 8.202.2 Define Documentation

#### 8.202.2.1 `#define DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID(METHOD)`

Value:

```
template <class Stat> \
    std::vector<Real> \
    SequenceStatistics<Stat>::METHOD() const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(); \
        return results_; \
    }
```

#### 8.202.2.2 `#define DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)`

Value:

```
template <class Stat> \
    std::vector<Real> \
    SequenceStatistics<Stat>::METHOD(Real x) const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(x); \
        return results_; \
    }
```

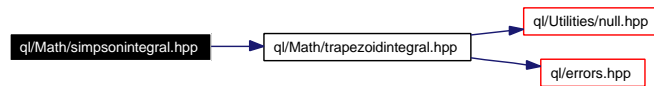
## 8.203 ql/Math/simpsonintegral.hpp File Reference

### 8.203.1 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.*

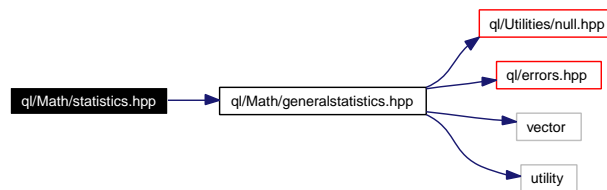
## 8.204 ql/Math/statistics.hpp File Reference

### 8.204.1 Detailed Description

statistics tool with risk measures

```
#include <ql/Math/generalstatistics.hpp>
```

Include dependency graph for statistics.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef GeneralStatistics [QuantLib::Statistics](#)  
*default statistics tool*

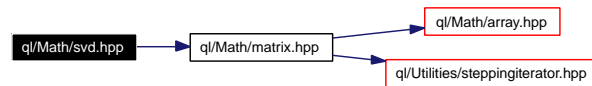
## 8.205 ql/Math/svd.hpp File Reference

### 8.205.1 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.*

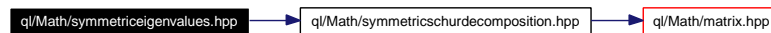
## 8.206 ql/Math/symmetriceigenvalues.hpp File Reference

### 8.206.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

```
#include <ql/Math/symmetricschurdecomposition.hpp>
```

Include dependency graph for symmetriceigenvalues.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- Disposable< Array > **QuantLib::SymmetricEigenvalues** (Matrix &s)
- Disposable< Matrix > **QuantLib::SymmetricEigenvectors** (Matrix &s)

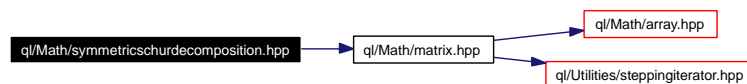
## 8.207 ql/Math/symmetricschurdecomposition.hpp File Reference

### 8.207.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for symmetricschurdecomposition.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SymmetricSchurDecomposition](#)  
*symmetric threshold Jacobi algorithm.*



## 8.208 ql/Math/tqreigendecomposition.hpp File Reference

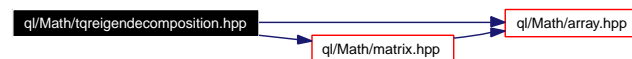
### 8.208.1 Detailed Description

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*

## 8.209 ql/Math/transformedgrid.hpp File Reference

### 8.209.1 Detailed Description

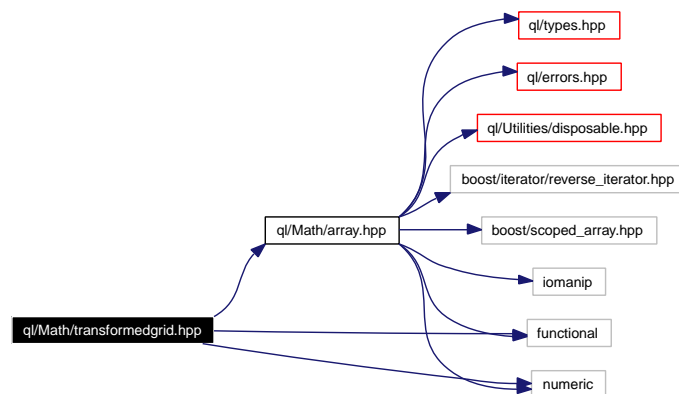
encapsulates a grid

```
#include <ql/Math/array.hpp>
```

```
#include <functional>
```

```
#include <numeric>
```

Include dependency graph for transformedgrid.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [TransformedGrid](#)  
*transformed grid*

## 8.210 ql/Math/trapezoidintegral.hpp File Reference

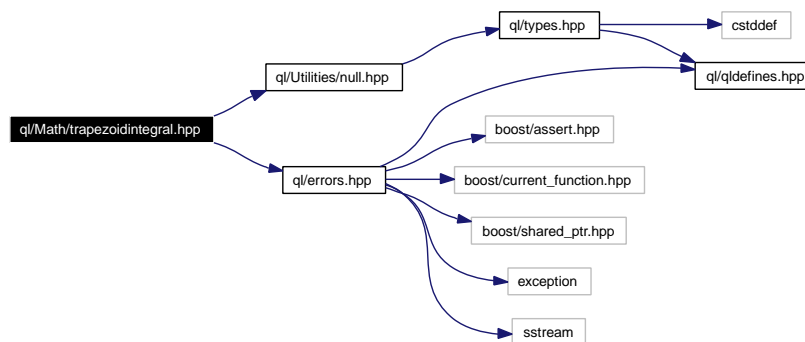
### 8.210.1 Detailed Description

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.*

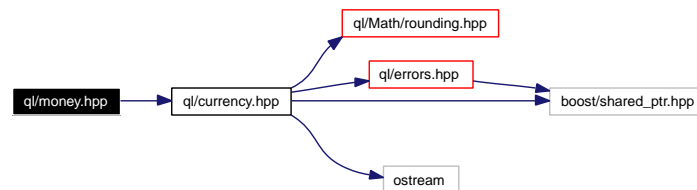
## 8.211 ql/money.hpp File Reference

### 8.211.1 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*

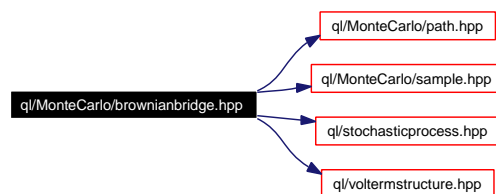
## 8.212 ql/MonteCarlo/brownianbridge.hpp File Reference

### 8.212.1 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.*

## 8.213 ql/MonteCarlo/getcovariance.hpp File Reference

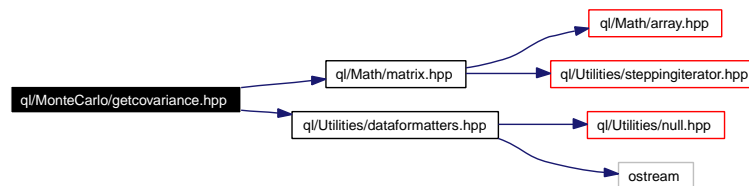
### 8.213.1 Detailed Description

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 > QuantLib::getCovariance (DataIterator volBegin, DataIterator volEnd, const Matrix &corr, Real tolerance=1.0e-12)`

## 8.214 ql/MonteCarlo/mctraits.hpp File Reference

### 8.214.1 Detailed Description

Monte Carlo policies.

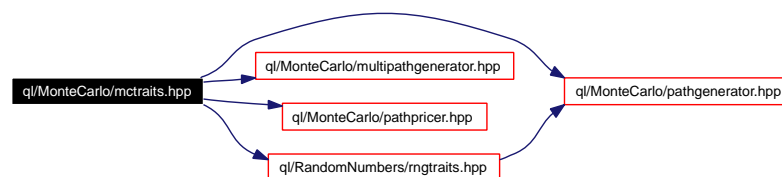
```
#include <ql/MonteCarlo/pathgenerator.hpp>
```

```
#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*

## 8.215 ql/MonteCarlo/mctypedefs.hpp File Reference

### 8.215.1 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 > >` [QuantLib::OneFactorMonteCarloOption](#)  
*default choice for one-factor Monte Carlo model.*
- typedef `MonteCarloModel< MultiVariate< PseudoRandom > >` [QuantLib::MultiFactorMonteCarloOption](#)  
*default choice for multi-factor Monte Carlo model.*



## 8.216 ql/MonteCarlo/montecarlomodel.hpp File Reference

### 8.216.1 Detailed Description

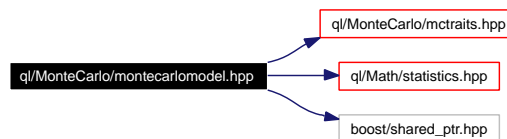
General purpose Monte Carlo model.

```
#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.*

## 8.217 ql/MonteCarlo/multipath.hpp File Reference

### 8.217.1 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.*

## 8.218 ql/MonteCarlo/multipathgenerator.hpp File Reference

### 8.218.1 Detailed Description

Generates a multi path from a random-array generator.

```
#include <ql/stochasticprocess.hpp>
```

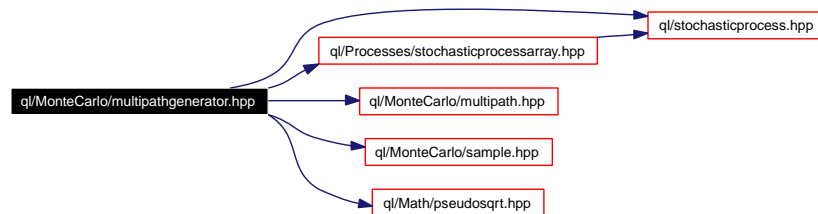
```
#include <ql/Processes/stochasticprocessarray.hpp>
```

```
#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.*

## 8.219 ql/MonteCarlo/path.hpp File Reference

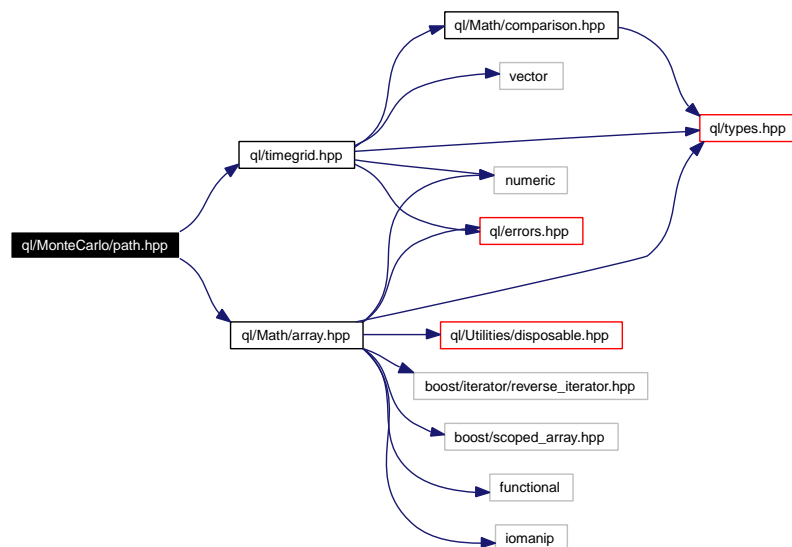
### 8.219.1 Detailed Description

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`

## 8.220 ql/MonteCarlo/pathgenerator.hpp File Reference

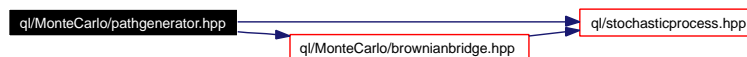
### 8.220.1 Detailed Description

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.*

## 8.221 ql/MonteCarlo/pathpricer.hpp File Reference

### 8.221.1 Detailed Description

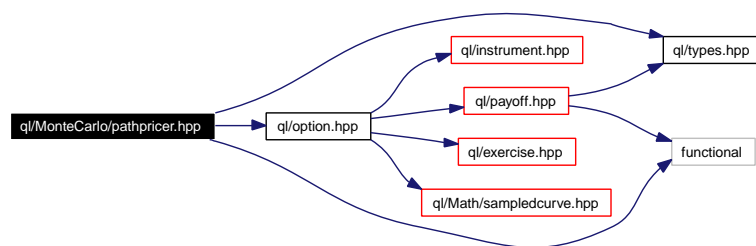
base class for single-path pricers

```
#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*

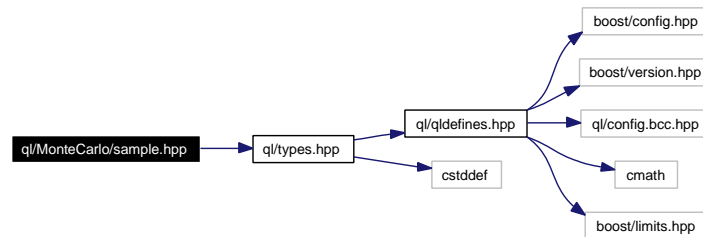
## 8.222 ql/MonteCarlo/sample.hpp File Reference

### 8.222.1 Detailed Description

weighted sample

```
#include <ql/types.hpp>
```

Include dependency graph for sample.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- struct [Sample](#)  
*weighted sample*

## 8.223 ql/numericalmethod.hpp File Reference

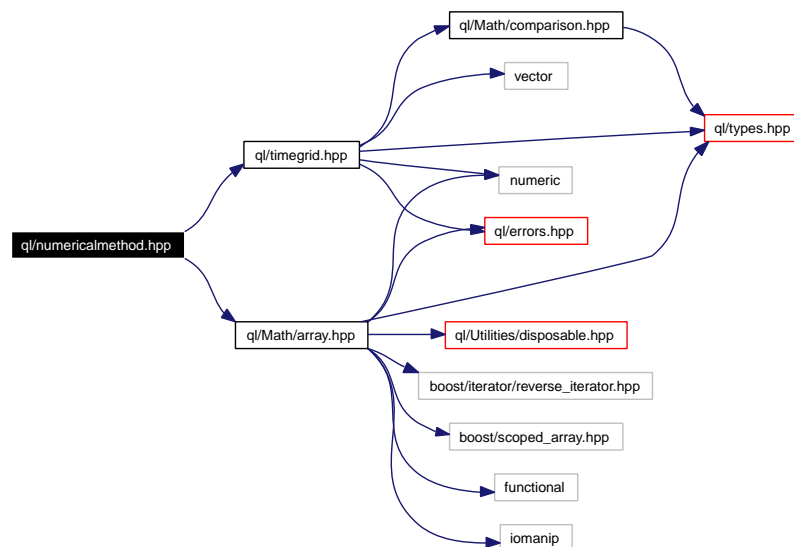
### 8.223.1 Detailed Description

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.*



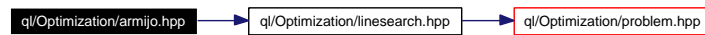
## 8.224 ql/Optimization/armijo.hpp File Reference

### 8.224.1 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.*

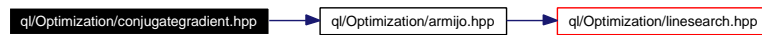
## 8.225 ql/Optimization/conjugategradient.hpp File Reference

### 8.225.1 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.*

## 8.226 ql/Optimization/constraint.hpp File Reference

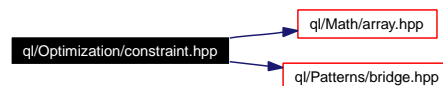
### 8.226.1 Detailed Description

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*

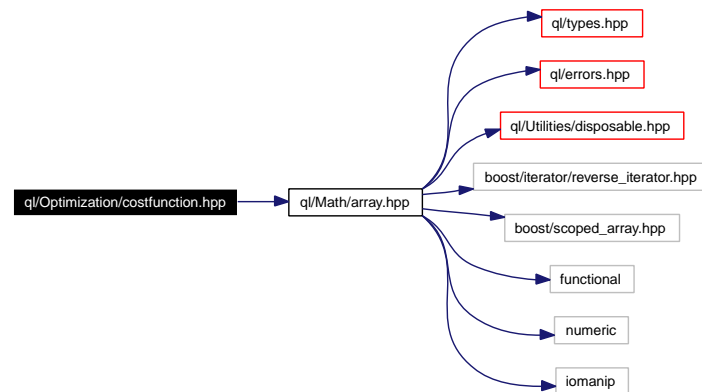
## 8.227 ql/Optimization/costfunction.hpp File Reference

### 8.227.1 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.*

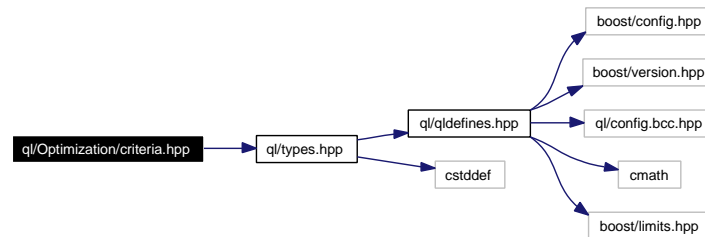
## 8.228 ql/Optimization/criteria.hpp File Reference

### 8.228.1 Detailed Description

Optimization criteria class.

```
#include <ql/types.hpp>
```

Include dependency graph for criteria.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [EndCriteria](#)  
*Criteria to end optimization process.*

## 8.229 ql/Optimization/leastsquare.hpp File Reference

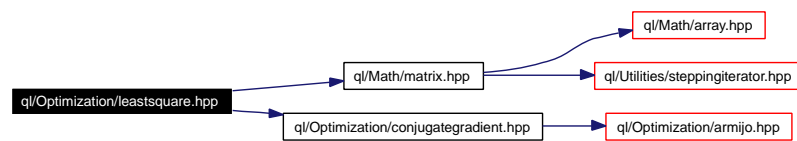
### 8.229.1 Detailed Description

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.*

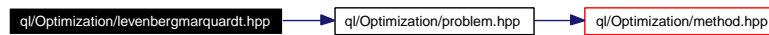
## 8.230 ql/Optimization/levenbergmarquardt.hpp File Reference

### 8.230.1 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.*

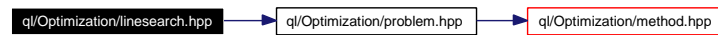
## 8.231 ql/Optimization/linesearch.hpp File Reference

### 8.231.1 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.*



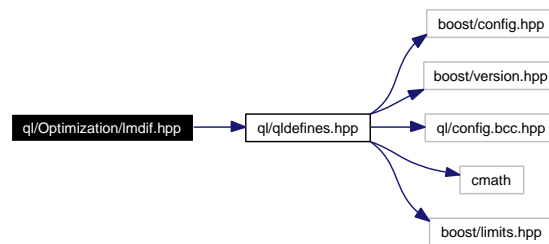
## 8.232 ql/Optimization/lmdif.hpp File Reference

### 8.232.1 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 **QuantLib::MINPACK::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 ldffac, int \*ipvt, double \*qtf, double \*wa1, double \*wa2, double \*wa3, double \*wa4)

## 8.233 ql/Optimization/method.hpp File Reference

### 8.233.1 Detailed Description

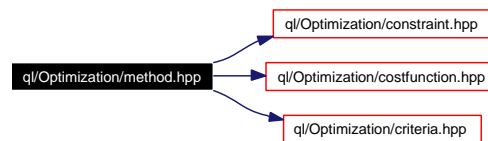
Abstract optimization method class.

```
#include <ql/Optimization/constraint.hpp>
```

```
#include <ql/Optimization/costfunction.hpp>
```

```
#include <ql/Optimization/criteria.hpp>
```

Include dependency graph for method.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [OptimizationMethod](#)  
*Abstract class for constrained optimization method.*

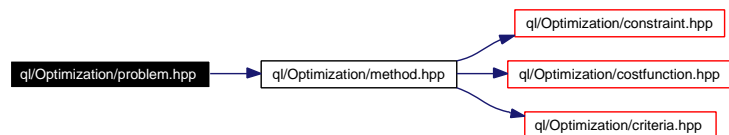
## 8.234 ql/Optimization/problem.hpp File Reference

### 8.234.1 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.*

## 8.235 ql/Optimization/simplex.hpp File Reference

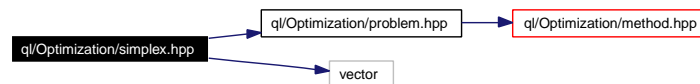
### 8.235.1 Detailed Description

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.*

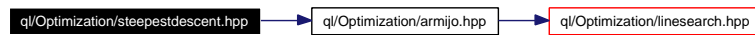
## 8.236 ql/Optimization/steepestdescent.hpp File Reference

### 8.236.1 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.*

## 8.237 ql/option.hpp File Reference

### 8.237.1 Detailed Description

Base option class.

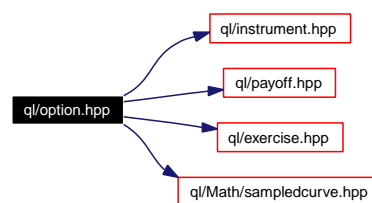
```
#include <ql/instrument.hpp>
```

```
#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*

### Functions

- `std::ostream & QuantLib::operator<< (std::ostream &out, Option::Type type)`

## 8.238 ql/Patterns/bridge.hpp File Reference

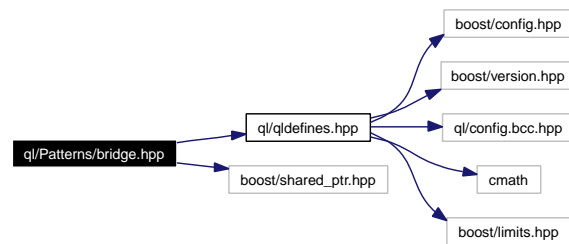
### 8.238.1 Detailed Description

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.*

## 8.239 ql/Patterns/composite.hpp File Reference

### 8.239.1 Detailed Description

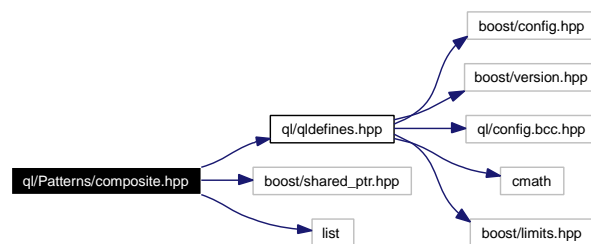
composite pattern

```
#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.*



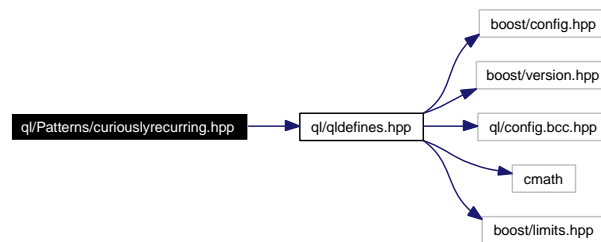
## 8.240 ql/Patterns/curiouslyrecurring.hpp File Reference

### 8.240.1 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.*

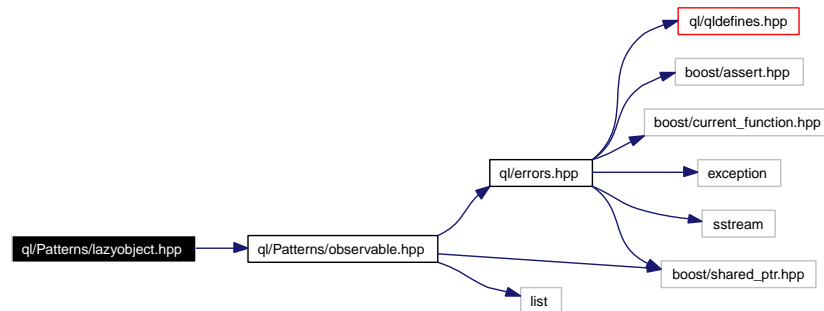
## 8.241 ql/Patterns/lazyobject.hpp File Reference

### 8.241.1 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.*

## 8.242 ql/Patterns/observable.hpp File Reference

### 8.242.1 Detailed Description

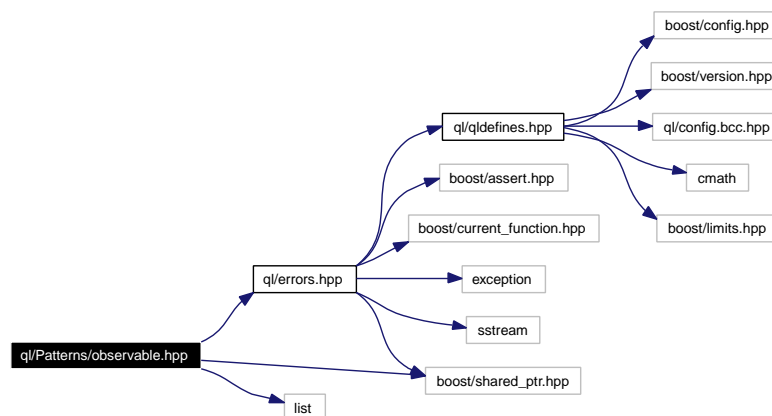
observer/observable pattern

```
#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.*

## 8.243 ql/Patterns/singleton.hpp File Reference

### 8.243.1 Detailed Description

basic support for the singleton pattern

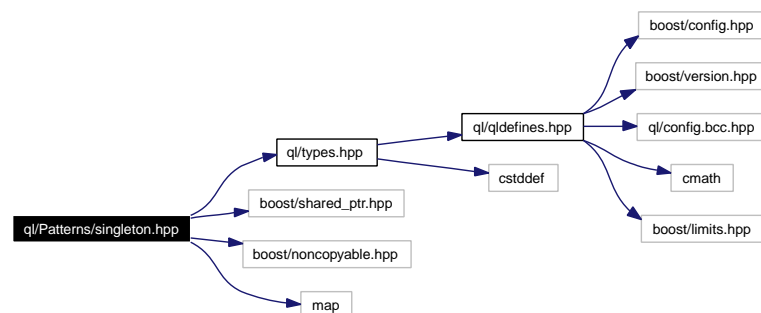
```
#include <ql/types.hpp>
```

```
#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.*

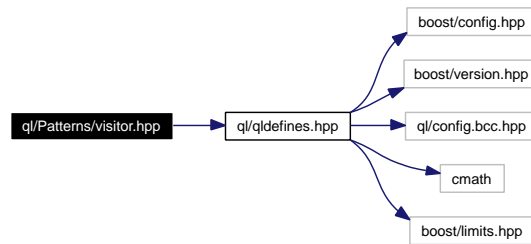
## 8.244 ql/Patterns/visitor.hpp File Reference

### 8.244.1 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*

## 8.245 ql/payoff.hpp File Reference

### 8.245.1 Detailed Description

Option payoff classes.

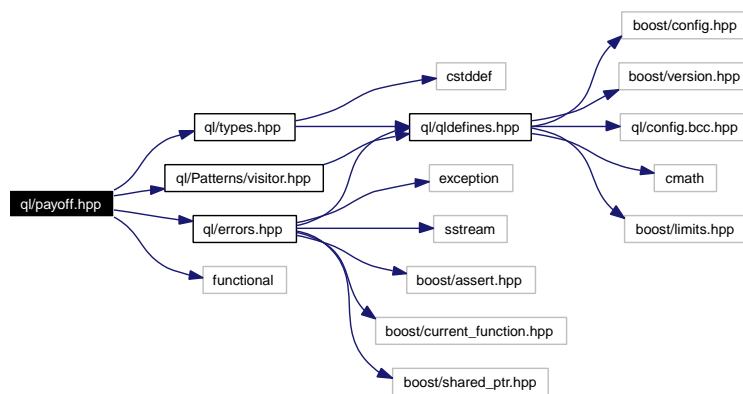
```
#include <ql/types.hpp>
```

```
#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.*

## 8.246 ql/Pricers/discretegeometricaso.hpp File Reference

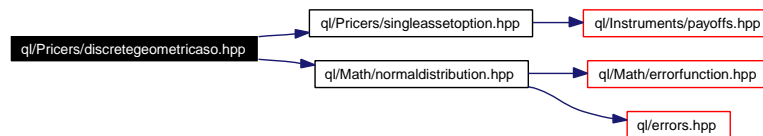
### 8.246.1 Detailed Description

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).*

## 8.247 ql/Pricers/mccliquestoption.hpp File Reference

### 8.247.1 Detailed Description

Cliquet option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
```

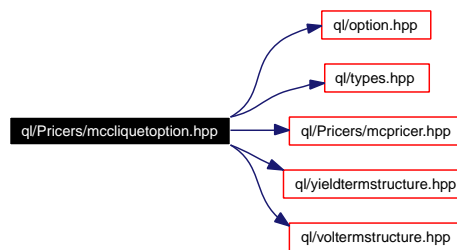
```
#include <ql/types.hpp>
```

```
#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*



## 8.248 ql/Pricers/mcdiscretearithmeticaso.hpp File Reference

### 8.248.1 Detailed Description

Discrete Arithmetic Average Strike Option.

```
#include <ql/option.hpp>
```

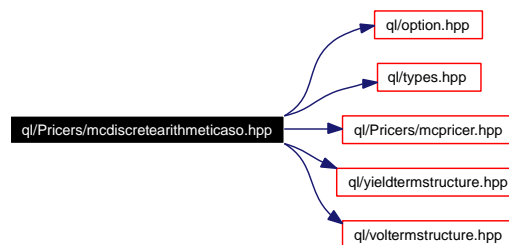
```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcdiscretearithmeticaso.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [McDiscreteArithmeticASO](#)  
*example of Monte Carlo pricer using a control variate.*

## 8.249 ql/Pricers/mceverest.hpp File Reference

### 8.249.1 Detailed Description

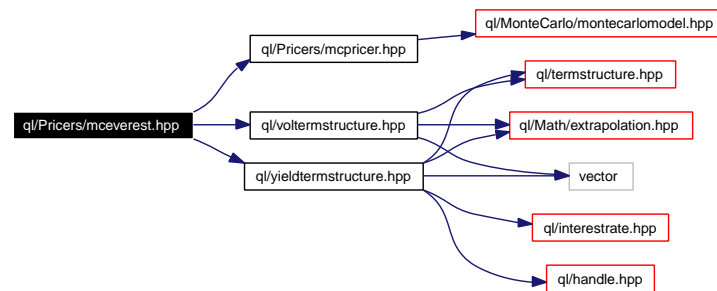
Everest-type option pricer

```
#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.*

## 8.250 ql/Pricers/mchimalaya.hpp File Reference

### 8.250.1 Detailed Description

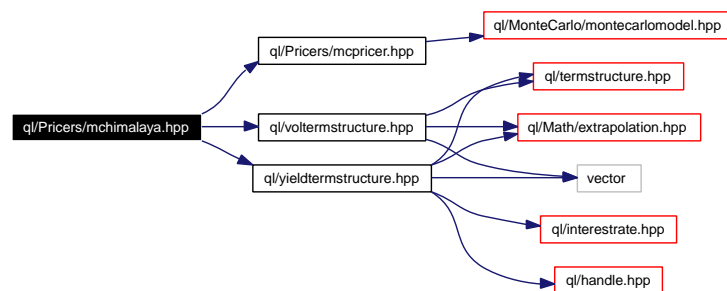
Himalayan-type option pricer.

```
#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.*

## 8.251 ql/Pricers/mcmaxbasket.hpp File Reference

### 8.251.1 Detailed Description

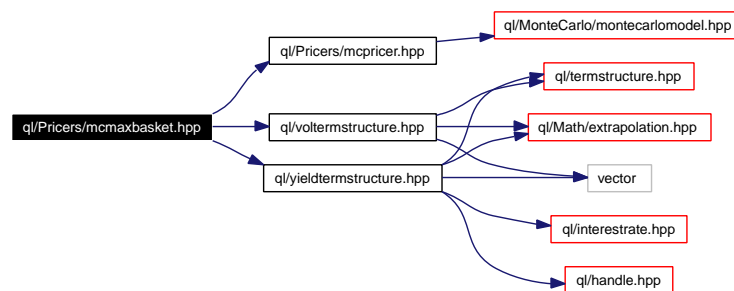
Max Basket Monte Carlo pricer.

```
#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*

## 8.252 ql/Pricers/mcpagoda.hpp File Reference

### 8.252.1 Detailed Description

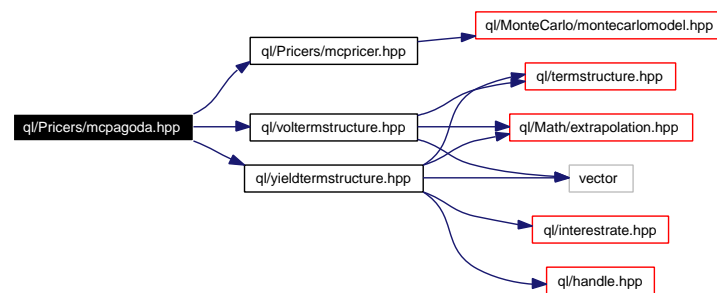
Roofed multi asset Asian option.

```
#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*

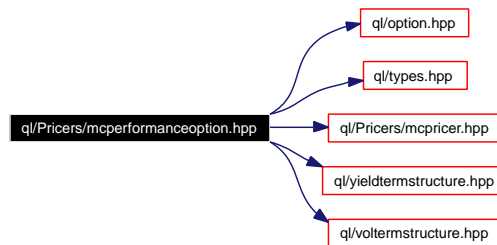
## 8.253 ql/Pricers/mcperformanceoption.hpp File Reference

### 8.253.1 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.*

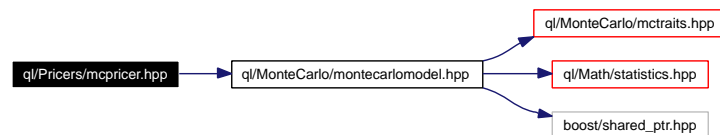
## 8.254 ql/Pricers/mcpricer.hpp File Reference

### 8.254.1 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*

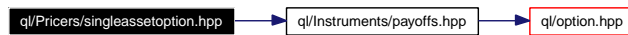
## 8.255 ql/Pricers/singleassetoption.hpp File Reference

### 8.255.1 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.*



## 8.256 ql/pricingengine.hpp File Reference

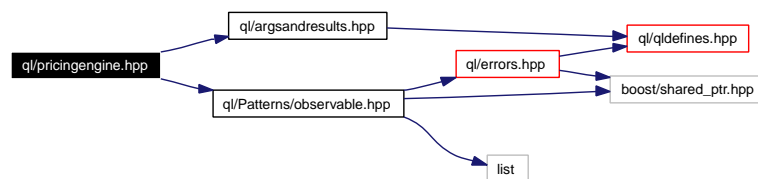
### 8.256.1 Detailed Description

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*

## 8.257 `ql/PricingEngines/americanpayoffatexpiry.hpp` File Reference

### 8.257.1 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`

## 8.258 ql/PricingEngines/americanpayoffathit.hpp File Reference

### 8.258.1 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](#)

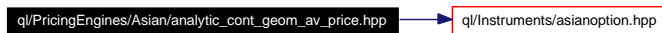
## 8.259 ql/PricingEngines/Asian/analytic\_cont\_geom\_av\_price.hpp File Reference

### 8.259.1 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.*

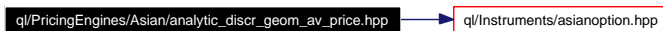
## 8.260 ql/PricingEngines/Asian/analytic\_discr\_geom\_av\_price.hpp File Reference

### 8.260.1 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.*

## 8.261 ql/PricingEngines/Asian/mc\_discr\_arith\_av\_price.hpp File Reference

### 8.261.1 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.*

## 8.262 ql/PricingEngines/Asian/mc\_discr\_geom\_av\_price.hpp File Reference

### 8.262.1 Detailed Description

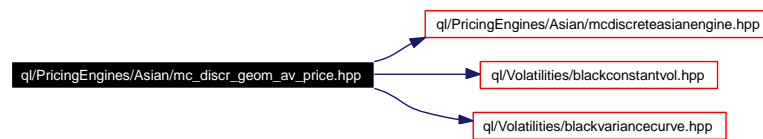
Monte Carlo engine for discrete geometric average price Asian.

```
#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.*

## 8.263 ql/PricingEngines/Asian/mcdiscreteasianengine.hpp File Reference

### 8.263.1 Detailed Description

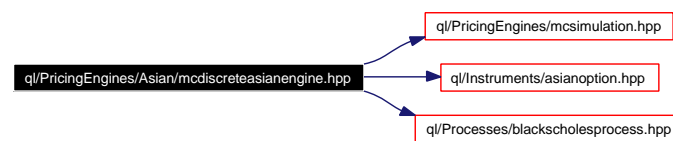
Monte Carlo pricing engine for discrete average Asians.

```
#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.*



## 8.264 ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference

### 8.264.1 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.*

## 8.265 ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference

### 8.265.1 Detailed Description

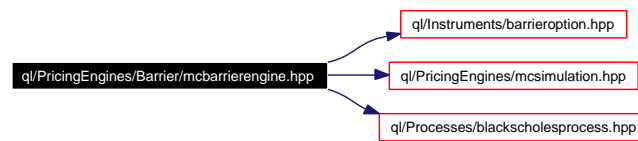
Monte Carlo barrier option engines.

```
#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.*

## 8.266 ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference

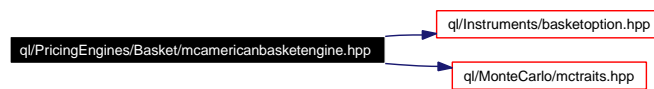
### 8.266.1 Detailed Description

Least-square Monte Carlo engines.

```
#include <ql/Instruments/basketoption.hpp>
```

```
#include <ql/MonteCarlo/mctraits.hpp>
```

Include dependency graph for mcamericanbasketengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [MCAmericanBasketEngine](#)  
*least-square Monte Carlo engine*

## 8.267 ql/PricingEngines/Basket/mcbasketengine.hpp File Reference

### 8.267.1 Detailed Description

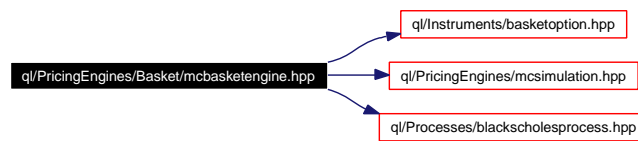
European basket MC Engine.

```
#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.*

## 8.268 ql/PricingEngines/Basket/stulzengine.hpp File Reference

### 8.268.1 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.*

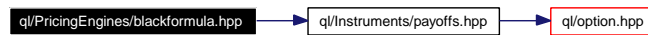
## 8.269 ql/PricingEngines/blackformula.hpp File Reference

### 8.269.1 Detailed Description

Black formula.

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for blackformula.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackFormula**  
*Black-formula calculator.*

## 8.270 ql/PricingEngines/blackmodel.hpp File Reference

### 8.270.1 Detailed Description

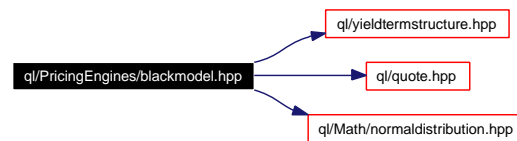
Abstract class for Black-type models (market models).

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/quote.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for blackmodel.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackModel**  
*Black-model for vanilla interest-rate derivatives.*

## 8.271 ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference

### 8.271.1 Detailed Description

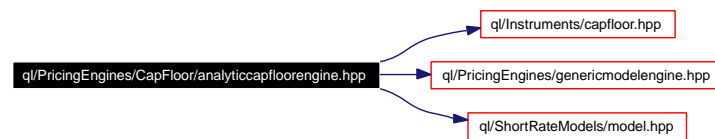
Analytic engine for caps/floors.

```
#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.*



## 8.272 ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference

### 8.272.1 Detailed Description

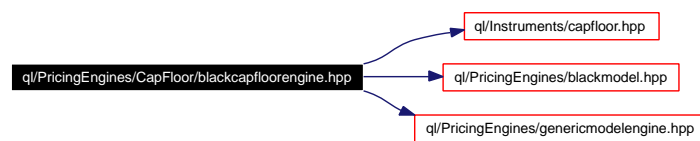
Black-formula cap/floor engine.

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/PricingEngines/blackmodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for blackcapfloorengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackCapFloorEngine**  
*Black-formula cap/floor engine.*

## 8.273 ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference

### 8.273.1 Detailed Description

discretized cap/floor

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for discretizedcapfloor.hpp:



## Namespaces

- namespace **QuantLib**

## 8.274 ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference

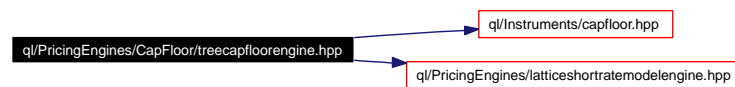
### 8.274.1 Detailed Description

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.*

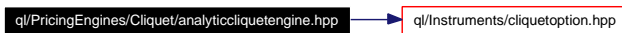
## 8.275 ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference

### 8.275.1 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.*

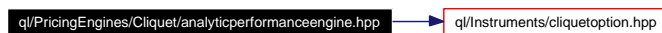
## 8.276 ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference

### 8.276.1 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.*

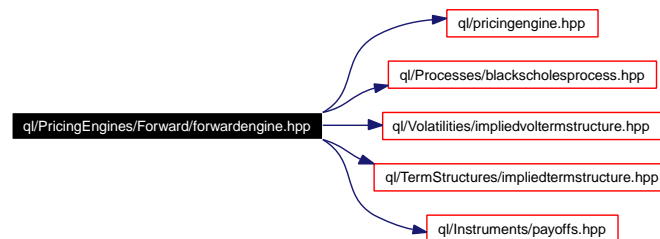
## 8.277 ql/PricingEngines/Forward/forwardengine.hpp File Reference

### 8.277.1 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.*

## 8.278 ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference

### 8.278.1 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.*

## 8.279 ql/PricingEngines/genericmodelengine.hpp File Reference

### 8.279.1 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.*



## 8.280 ql/PricingEngines/greeks.hpp File Reference

### 8.280.1 Detailed Description

default greek calculations

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for greeks.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- [Real QuantLib::blackScholesTheta](#) (const boost::shared\_ptr< BlackScholesProcess > &, [Real](#) value, [Real](#) delta, [Real](#) gamma)  
*default theta calculation for Black-Scholes options*
- [Real QuantLib::defaultThetaPerDay](#) ([Real](#) theta)  
*default theta-per-day calculation*

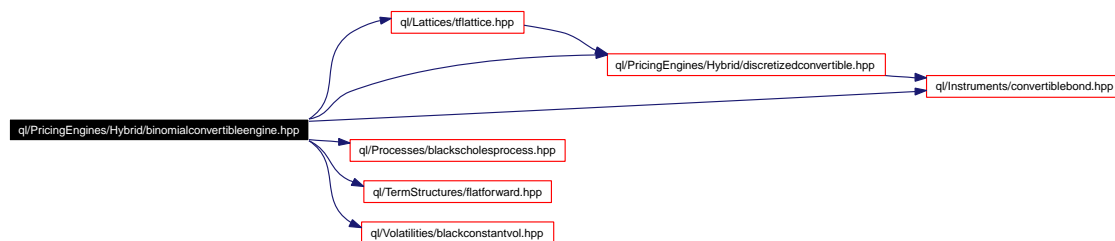
## 8.281 ql/PricingEngines/Hybrid/binomialconvertibleengine.hpp File Reference

### 8.281.1 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.*

## 8.282 ql/PricingEngines/Hybrid/discretizedconvertible.hpp File Reference

### 8.282.1 Detailed Description

discretized convertible

```
#include <ql/discretizedasset.hpp>
```

```
#include <ql/Instruments/convertiblebond.hpp>
```

Include dependency graph for discretizedconvertible.hpp:



### Namespaces

- namespace **QuantLib**

## 8.283 ql/PricingEngines/latticeshortratemodelengine.hpp File Reference

### 8.283.1 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.*

## 8.284 ql/PricingEngines/mcsimulation.hpp File Reference

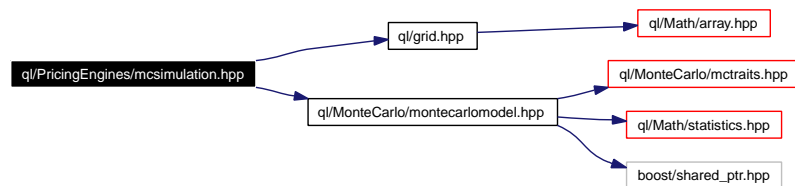
### 8.284.1 Detailed Description

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*

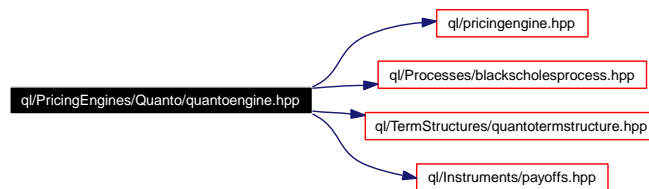
## 8.285 ql/PricingEngines/Quanto/quantoengine.hpp File Reference

### 8.285.1 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.*

## 8.286 ql/PricingEngines/Swaption/blackswaptionengine.hpp File Reference

### 8.286.1 Detailed Description

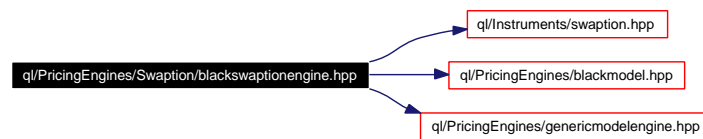
Black-formula swaption engine.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/PricingEngines/blackmodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for blackswaptionengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class **BlackSwaptionEngine**  
*Black-formula swaption engine.*

## 8.287 ql/PricingEngines/Swaption/discretizedswaption.hpp File Reference

### 8.287.1 Detailed Description

Discretized swaption class.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for discretizedswaption.hpp:



## Namespaces

- namespace **QuantLib**



## 8.288 ql/PricingEngines/SwapTION/g2swaptionengine.hpp File Reference

### 8.288.1 Detailed Description

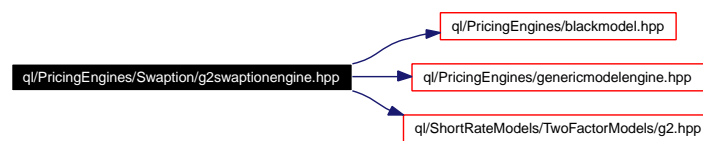
SwapTION pricing engine for two-factor additive Gaussian Model G2++.

```
#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*

## 8.289 ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp File Reference

### 8.289.1 Detailed Description

Swaption engine using Jamshidian's decomposition.

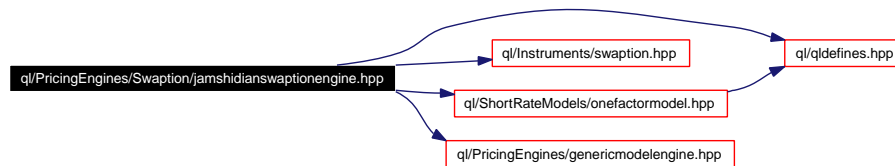
```
#include <ql/qldefines.hpp>
```

```
#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.*

## 8.290 ql/PricingEngines/Swaption/lfmwaptionengine.hpp File Reference

### 8.290.1 Detailed Description

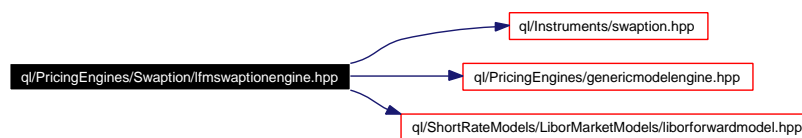
libor forward model swaption engine based on black formula

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

```
#include <ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp>
```

Include dependency graph for lfmwaptionengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LfmSwaptionEngine](#)  
*libor forward model swaption engine based on black formula*

## 8.291 ql/PricingEngines/Swaption/treeswaptionengine.hpp File Reference

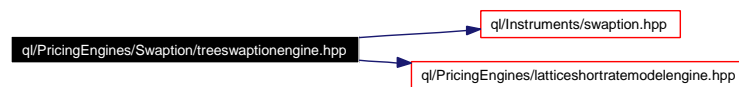
### 8.291.1 Detailed Description

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.*

### Typedefs

- typedef [TreeVanillaSwapEngine](#) [QuantLib::TreeSimpleSwapEngine](#)

## 8.292 ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference

### 8.292.1 Detailed Description

analytic digital American option engine

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for analyticdigitalamericanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AnalyticDigitalAmericanEngine](#)

## 8.293 ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp File Reference

### 8.293.1 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.*

## 8.294 ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference

### 8.294.1 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.*

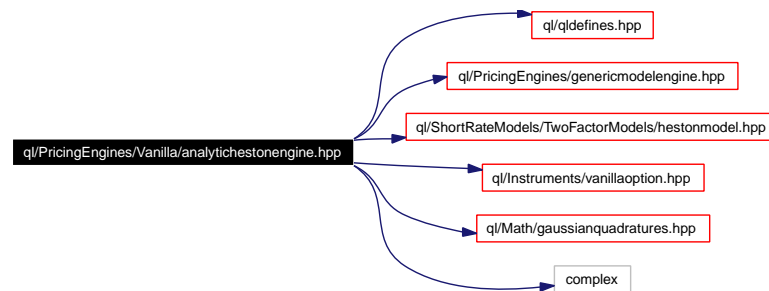
## 8.295 ql/PricingEngines/Vanilla/analytichestonengine.hpp File Reference

### 8.295.1 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*



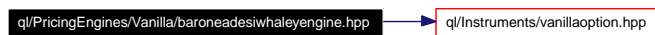
## 8.296 ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference

### 8.296.1 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](#)

## 8.297 ql/PricingEngines/Vanilla/batesengine.hpp File Reference

### 8.297.1 Detailed Description

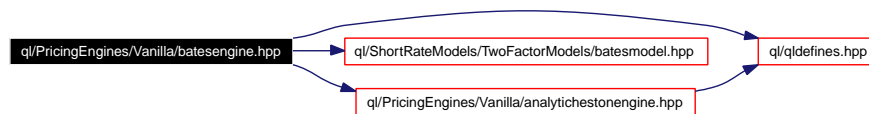
analytic Bates model engine

```
#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.*

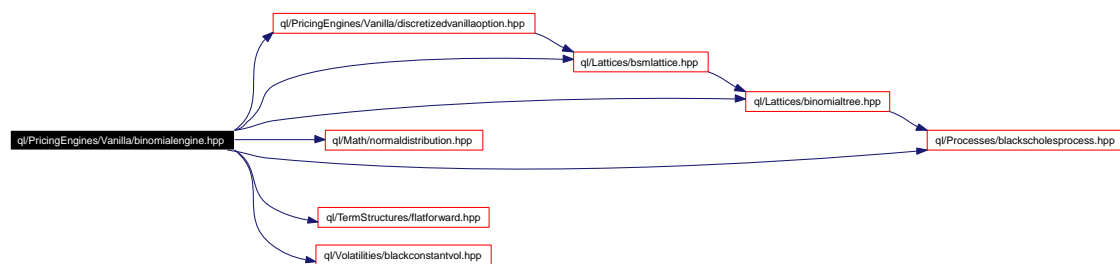
## 8.298 ql/PricingEngines/Vanilla/binomialengine.hpp File Reference

### 8.298.1 Detailed Description

Binomial option engine.

```
#include <ql/Lattices/binomialtree.hpp>
#include <ql/Lattices/bsmlattice.hpp>
#include <ql/Math/normaldistribution.hpp>
#include <ql/PricingEngines/Vanilla/discretizedvanillaoption.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.*

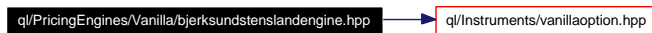
## 8.299 ql/PricingEngines/Vanilla/bjersundstenslandengine.hpp File Reference

### 8.299.1 Detailed Description

Bjersund and Stensland approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for bjersundstenslandengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BjersundStenslandApproximationEngine](#)

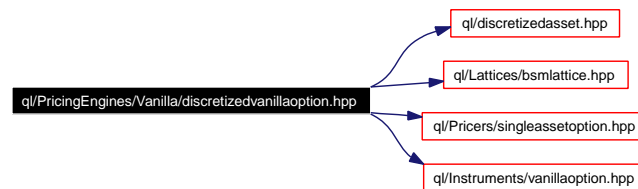
## 8.300 ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference

### 8.300.1 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**

## 8.301 ql/PricingEngines/Vanilla/fdamericanengine.hpp File Reference

### 8.301.1 Detailed Description

Finite-differences American option engine.

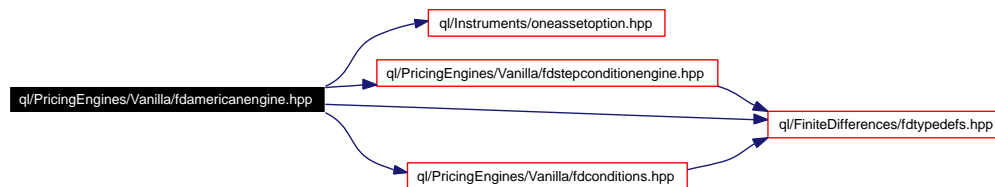
```
#include <ql/Instruments/oneassetoption.hpp>
```

```
#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 >` [QuantLib::FDAmericanEngine](#)

*Finite-differences pricing engine for American one asset options.*

## 8.302 ql/PricingEngines/Vanilla/fdbermudanengine.hpp File Reference

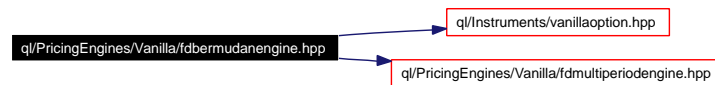
### 8.302.1 Detailed Description

finite-difference Bermudan engine

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdmultipleriodengine.hpp>
```

Include dependency graph for fdbermudanengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FDBermudanEngine](#)  
*Finite-differences Bermudan engine.*

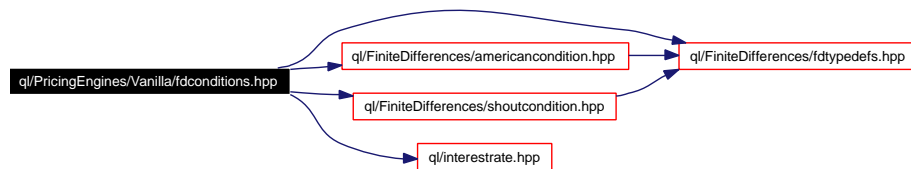
## 8.303 ql/PricingEngines/Vanilla/fdconditions.hpp File Reference

### 8.303.1 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/interestrates.hpp>
```

Include dependency graph for fdconditions.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FDAmericanCondition](#)



## 8.304 ql/PricingEngines/Vanilla/fddividendamericanengine.hpp File Reference

### 8.304.1 Detailed Description

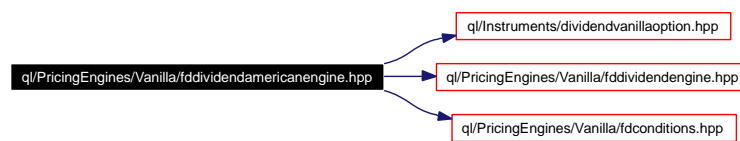
american engine with discrete deterministic dividends

```
#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 >` [QuantLib::FDDividendAmericanEngine](#)  
*Finite-differences pricing engine for dividend American options.*
- typedef `FDEngineAdapter< FDAmericanCondition< FDDividendEngineMerton73 >, DividendVanillaOption::engine >` **QuantLib::FDDividendAmericanEngineMerton73**
- typedef `FDEngineAdapter< FDAmericanCondition< FDDividendEngineShiftScale >, DividendVanillaOption::engine >` **QuantLib::FDDividendAmericanEngineShiftScale**

## 8.305 ql/PricingEngines/Vanilla/fddividendengine.hpp File Reference

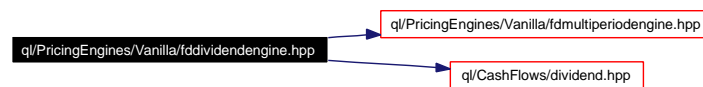
### 8.305.1 Detailed Description

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` **QuantLib::FDDividendEngine**

## 8.306 ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp File Reference

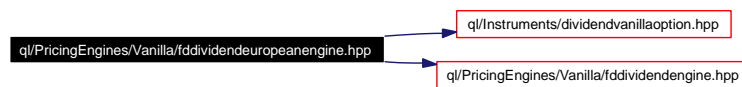
### 8.306.1 Detailed Description

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 `FDEngineAdapter< FDDividendEngine, DividendVanillaOption::engine >` [QuantLib::FDDividendEuropeanEngine](#)  
*Finite-differences pricing engine for dividend European options.*
- typedef `FDEngineAdapter< FDDividendEngineMerton73, DividendVanillaOption::engine >` **QuantLib::FDDividendEuropeanEngineMerton73**
- typedef `FDEngineAdapter< FDDividendEngineShiftScale, DividendVanillaOption::engine >` **QuantLib::FDDividendEuropeanEngineShiftScale**

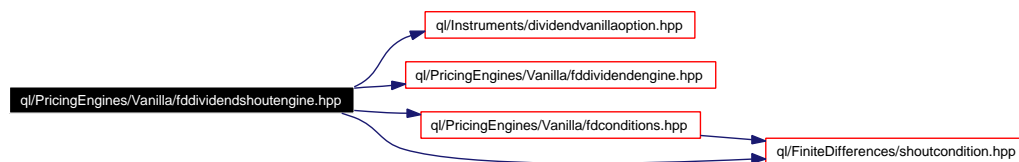
## 8.307 ql/PricingEngines/Vanilla/fddividendshoutengine.hpp File Reference

### 8.307.1 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 > **QuantLib::FDDividendShoutEngine**  
*Finite-differences shout engine with dividends.*
- typedef FDEngineAdapter< FDS shoutCondition< FDDividendEngineMerton73 >, DividendVanillaOption::engine > **QuantLib::FDDividendShoutEngineMerton73**
- typedef FDEngineAdapter< FDS shoutCondition< FDDividendEngineShiftScale >, DividendVanillaOption::engine > **QuantLib::FDDividendShoutEngineShiftScale**

## 8.308 ql/PricingEngines/Vanilla/fdeuropeanengine.hpp File Reference

### 8.308.1 Detailed Description

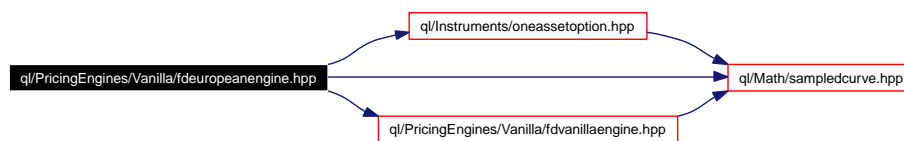
Finite-difference European engine.

```
#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.*

## 8.309 ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp File Reference

### 8.309.1 Detailed Description

base engine for options with events happening at specific times

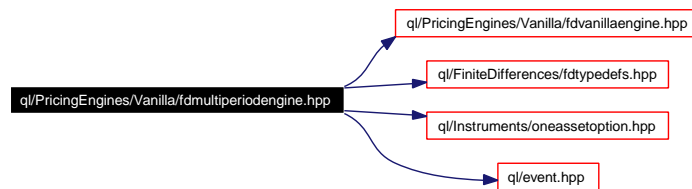
```
#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/Instruments/oneassetoption.hpp>
```

```
#include <ql/event.hpp>
```

Include dependency graph for fdmultiperiodengine.hpp:



### Namespaces

- namespace **QuantLib**

## 8.310 ql/PricingEngines/Vanilla/fdshoutengine.hpp File Reference

### 8.310.1 Detailed Description

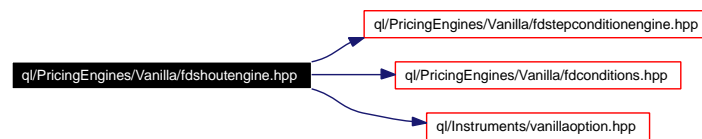
Finite-differences shout engine.

```
#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 >` [QuantLib::FDShoutEngine](#)

*Finite-differences pricing engine for shout vanilla options.*

## 8.311 ql/PricingEngines/Vanilla/fdstepconditionengine.hpp File Reference

### 8.311.1 Detailed Description

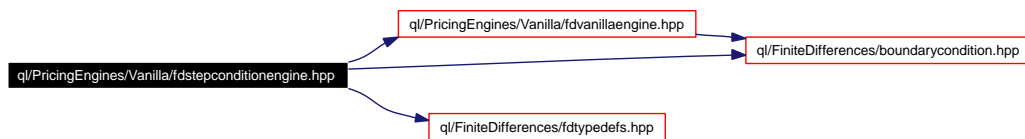
Finite-differences step-condition engine.

```
#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.*



## 8.312 ql/PricingEngines/Vanilla/fdvanillaengine.hpp File Reference

### 8.312.1 Detailed Description

Finite-differences vanilla-option engine.

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

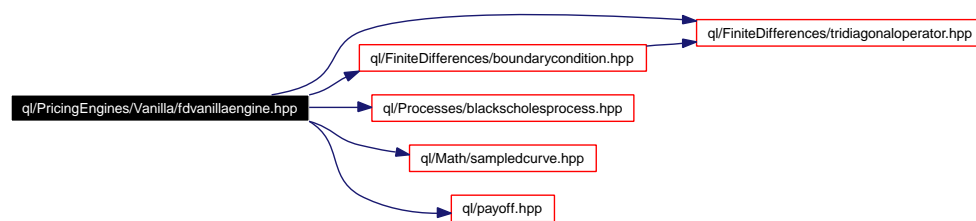
```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

```
#include <ql/Math/sampledcurve.hpp>
```

```
#include <ql/payoff.hpp>
```

Include dependency graph for fdvanillaengine.hpp:



### Namespaces

- namespace **QuantLib**

## 8.313 ql/PricingEngines/Vanilla/integralengine.hpp File Reference

### 8.313.1 Detailed Description

Integral option engine.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for integralengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [IntegralEngine](#)

## 8.314 ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference

### 8.314.1 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.*

## 8.315 ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference

### 8.315.1 Detailed Description

Ju quadratic (1999) approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for juquadraticengine.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [JuQuadraticApproximationEngine](#)

## 8.316 ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference

### 8.316.1 Detailed Description

digital option Monte Carlo engine

```
#include <ql/exercise.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

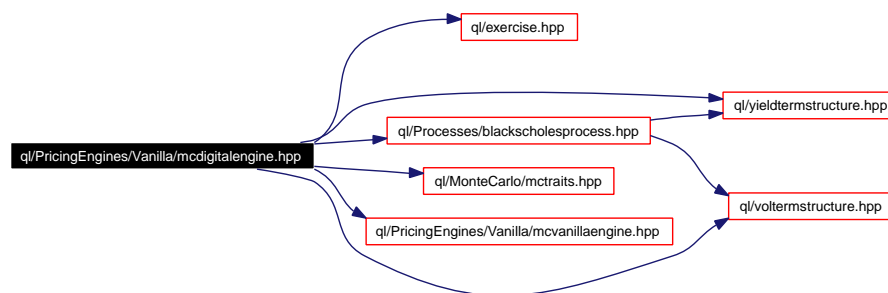
```
#include <ql/voltermstructure.hpp>
```

```
#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.*

## 8.317 ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference

### 8.317.1 Detailed Description

Monte Carlo European option engine.

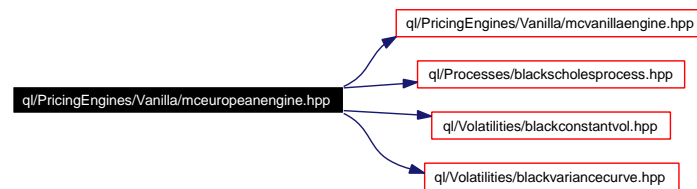
```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

```
#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.*

## 8.318 ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp File Reference

### 8.318.1 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.*

## 8.319 ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference

### 8.319.1 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.*



## 8.320 ql/Processes/blackscholesprocess.hpp File Reference

### 8.320.1 Detailed Description

Black-Scholes processes.

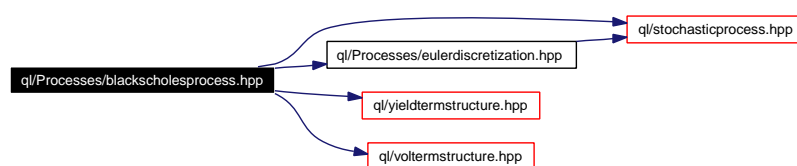
```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/Processes/eulerdiscretization.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for blackscholesprocess.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [BlackScholesProcess](#)  
*Black-Scholes stochastic process.*

## 8.321 ql/Processes/eulerdiscretization.hpp File Reference

### 8.321.1 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.*

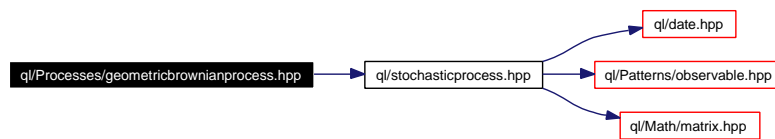
## 8.322 ql/Processes/geometricbrownianprocess.hpp File Reference

### 8.322.1 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.*

## 8.323 ql/Processes/hestonprocess.hpp File Reference

### 8.323.1 Detailed Description

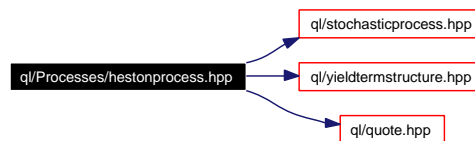
Heston stochastic process.

```
#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.*

## 8.324 ql/Processes/lfmcovarParams.hpp File Reference

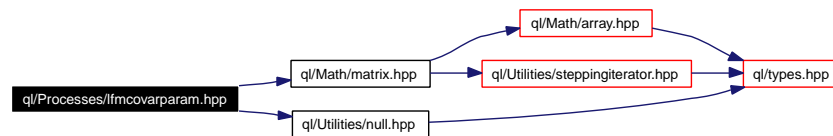
### 8.324.1 Detailed Description

volatility & correlation function for libor forward model process

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

Include dependency graph for lfmcovarParams.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LfmCovarianceParameterization](#)  
*libor market model parameterization*

## 8.325 ql/Processes/lfmhullwhiteparam.hpp File Reference

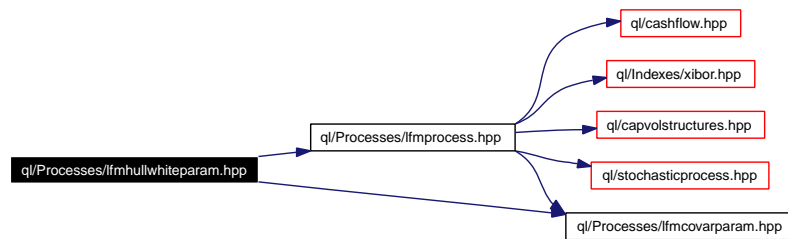
### 8.325.1 Detailed Description

libor market model parameterization based on Hull White

```
#include <ql/Processes/lfmprocess.hpp>
```

```
#include <ql/Processes/lfmcovarParams.hpp>
```

Include dependency graph for lfmhullwhiteparam.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LfmHullWhiteParameterization](#)

*libor market model parameterization based on Hull White paper*

## 8.326 ql/Processes/lfmprocess.hpp File Reference

### 8.326.1 Detailed Description

stochastic process of a libor forward model

```
#include <ql/cashflow.hpp>
```

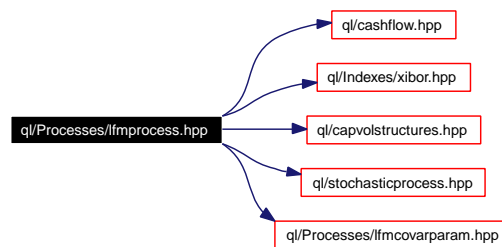
```
#include <ql/Indexes/xibor.hpp>
```

```
#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*

## 8.327 ql/Processes/merton76process.hpp File Reference

### 8.327.1 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.*



## 8.328 ql/Processes/ornsteinuhlenbeckprocess.hpp File Reference

### 8.328.1 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.*

## 8.329 ql/Processes/squarerootprocess.hpp File Reference

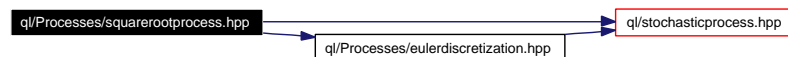
### 8.329.1 Detailed Description

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.*

## 8.330 ql/Processes/stochasticprocessarray.hpp File Reference

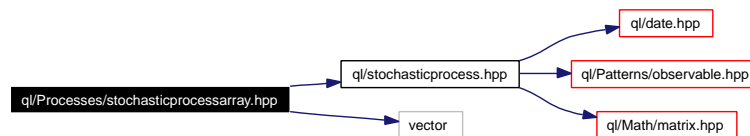
### 8.330.1 Detailed Description

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.*

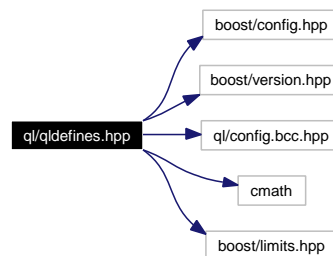
## 8.331 ql/qldefines.hpp File Reference

### 8.331.1 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** "0.3.12"  
*version string*
- #define **QL\_HEX\_VERSION** 0x000312f0  
*version hexadecimal number*
- #define **QL\_LIB\_VERSION** "0\_3\_12"  
*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](#)

## 8.332 ql/quote.hpp File Reference

### 8.332.1 Detailed Description

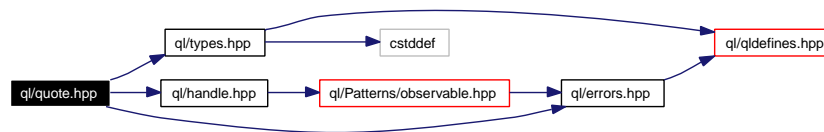
purely virtual base class for market observables

```
#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*

## 8.333 ql/RandomNumbers/boxmullergaussianrng.hpp File Reference

### 8.333.1 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.*

## 8.334 ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference

### 8.334.1 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.*



## 8.335 ql/RandomNumbers/faurersg.hpp File Reference

### 8.335.1 Detailed Description

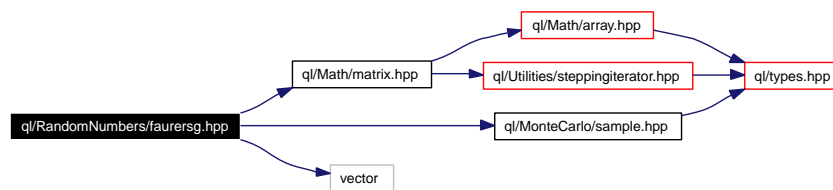
Faure low-discrepancy sequence generator.

```
#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.*

## 8.336 ql/RandomNumbers/haltonrsg.hpp File Reference

### 8.336.1 Detailed Description

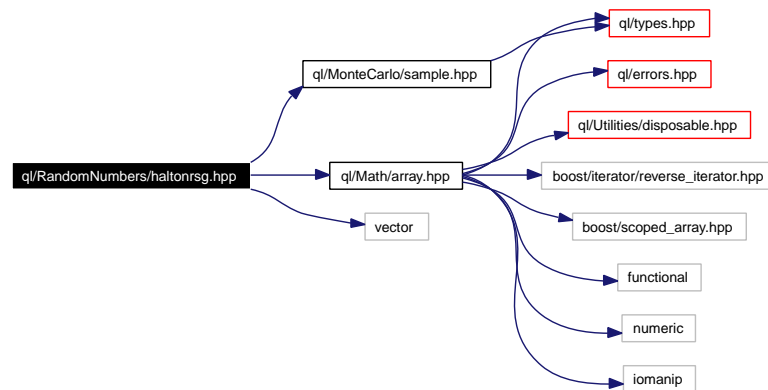
Halton low-discrepancy sequence generator.

```
#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.*

## 8.337 ql/RandomNumbers/inversecumulativerng.hpp File Reference

### 8.337.1 Detailed Description

Inverse cumulative Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for inversecumulativerng.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InverseCumulativeRng](#)  
*Inverse cumulative random number generator.*

## 8.338 ql/RandomNumbers/inversecumulativergs.hpp File Reference

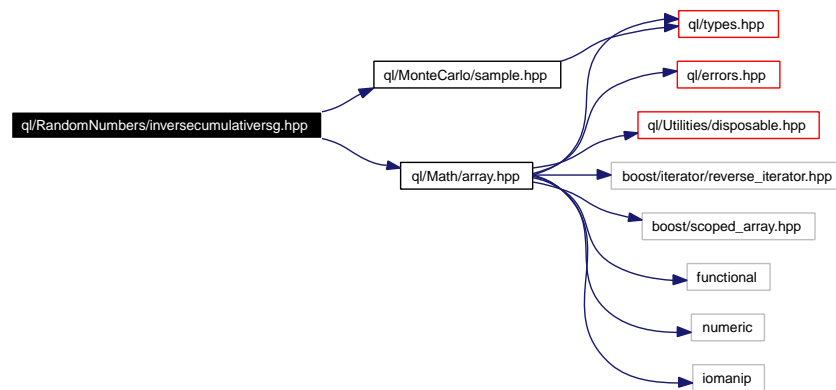
### 8.338.1 Detailed Description

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.*

## 8.339 ql/RandomNumbers/knuthuniformrng.hpp File Reference

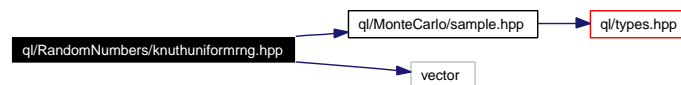
### 8.339.1 Detailed Description

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.*

## 8.340 ql/RandomNumbers/lecuyeruniformrng.hpp File Reference

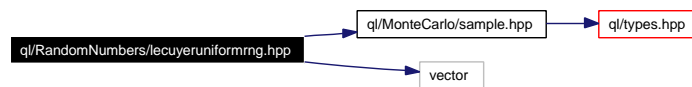
### 8.340.1 Detailed Description

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.*

## 8.341 ql/RandomNumbers/mt19937uniformrng.hpp File Reference

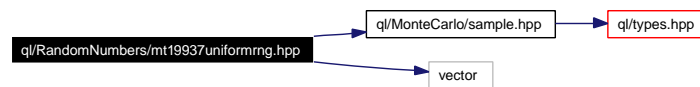
### 8.341.1 Detailed Description

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.*

## 8.342 ql/RandomNumbers/randomizedlds.hpp File Reference

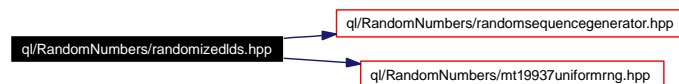
### 8.342.1 Detailed Description

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.*



## 8.343 ql/RandomNumbers/randomsequencegenerator.hpp File Reference

### 8.343.1 Detailed Description

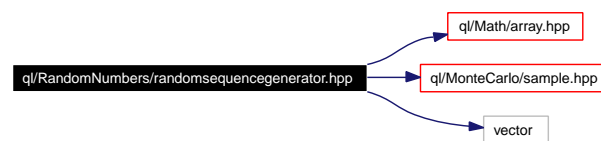
Random sequence generator based on a pseudo-random number generator.

```
#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.*

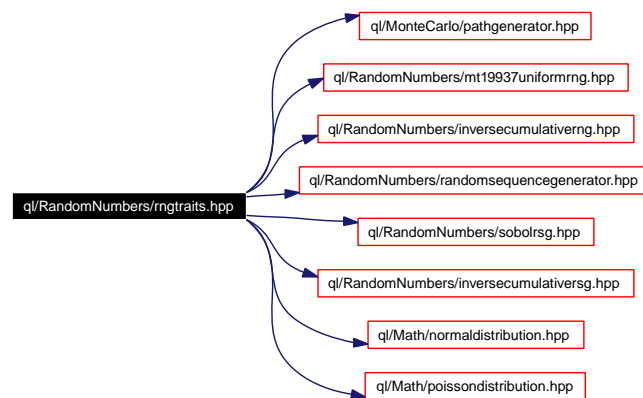
## 8.344 ql/RandomNumbers/rngtraits.hpp File Reference

### 8.344.1 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/inversecumulativersg.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 > [QuantLib::PseudoRandom](#)  
*default traits for pseudo-random number generation*
- typedef GenericPseudoRandom< MersenneTwisterUniformRng, InverseCumulativePoisson > [QuantLib::PoissonPseudoRandom](#)  
*traits for Poisson-distributed pseudo-random number generation*
- typedef GenericLowDiscrepancy< SobolRsg, InverseCumulativeNormal > [QuantLib::LowDiscrepancy](#)  
*default traits for low-discrepancy sequence generation*



## 8.345 ql/RandomNumbers/seedgenerator.hpp File Reference

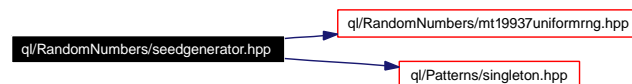
### 8.345.1 Detailed Description

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.*

## 8.346 ql/RandomNumbers/sobolrsg.hpp File Reference

### 8.346.1 Detailed Description

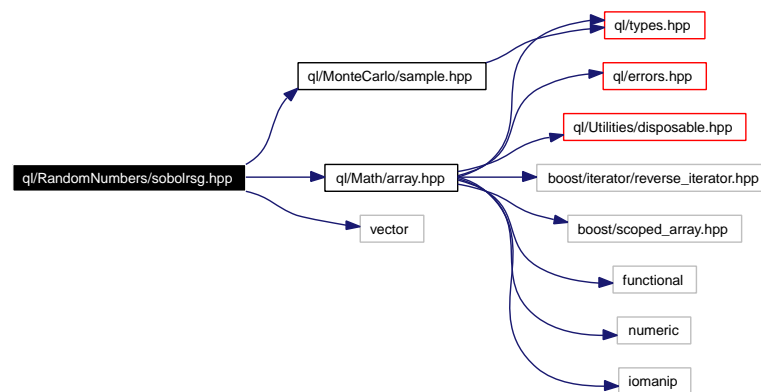
Sobol low-discrepancy sequence generator.

```
#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.*

## 8.347 ql/schedule.hpp File Reference

### 8.347.1 Detailed Description

date schedule

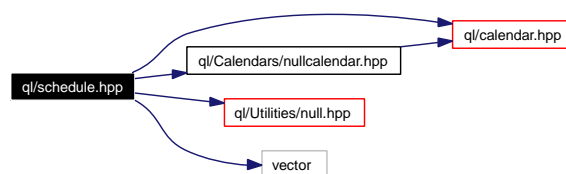
```
#include <ql/calendar.hpp>
```

```
#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*

## 8.348 ql/settings.hpp File Reference

### 8.348.1 Detailed Description

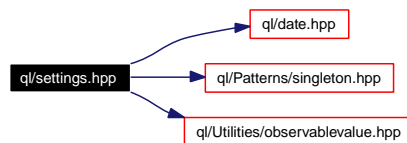
global repository for run-time library settings

```
#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 & QuantLib::operator<< (std::ostream &out, const Settings::DateProxy &p)`

## 8.349 ql/ShortRateModels/calibrationhelper.hpp File Reference

### 8.349.1 Detailed Description

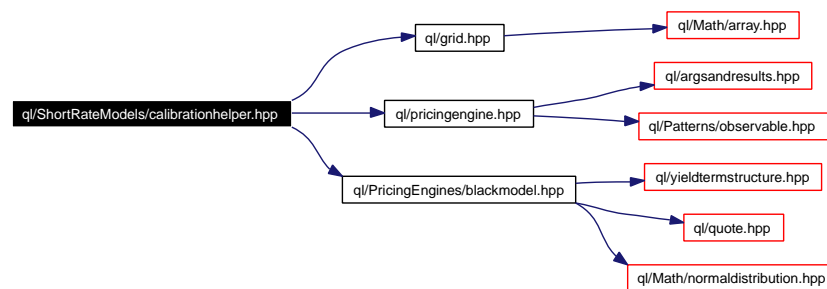
Calibration helper class.

```
#include <ql/grid.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*



## 8.350 ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference

### 8.350.1 Detailed Description

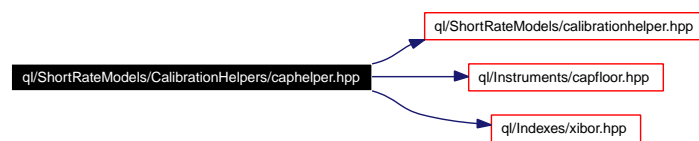
CapHelper calibration helper.

```
#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*

## 8.351 ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp File Reference

### 8.351.1 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*

## 8.352 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference

### 8.352.1 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*

## 8.353 ql/ShortRateModels/LiborMarketModels/lfmcovarproxy.hpp File Reference

### 8.353.1 Detailed Description

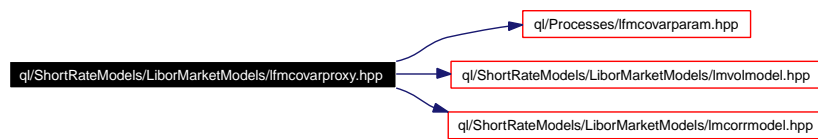
proxy for libor forward covariance parameterization

```
#include <ql/Processes/lfmcovarparam.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*

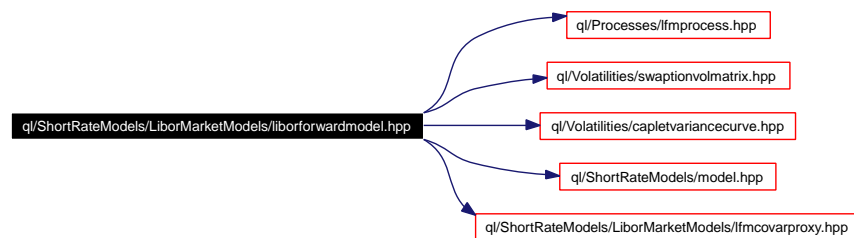
## 8.354 ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp File Reference

### 8.354.1 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.*

## 8.355 ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp File Reference

### 8.355.1 Detailed Description

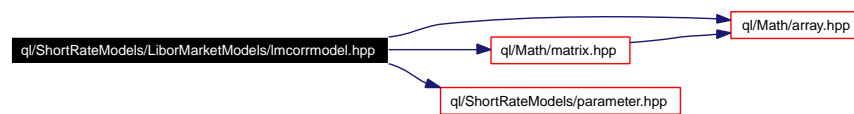
correlation model for libor market models

```
#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*

## 8.356 ql/ShortRateModels/LiborMarketModels/lmexpcorrmodel.hpp File Reference

### 8.356.1 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*

## 8.357 ql/ShortRateModels/LiborMarketModels/lmfixedvolmodel.hpp File Reference

### 8.357.1 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**



## 8.358 ql/ShortRateModels/LiborMarketModels/lmlinexpvolmodel.hpp File Reference

### 8.358.1 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*

## 8.359 ql/ShortRateModels/LiborMarketModels/Lmvolmodel.hpp File Reference

### 8.359.1 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*

## 8.360 ql/ShortRateModels/model.hpp File Reference

### 8.360.1 Detailed Description

Abstract interest rate model class.

```
#include <ql/option.hpp>
```

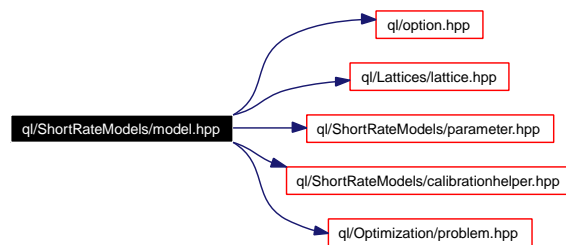
```
#include <ql/Lattices/lattice.hpp>
```

```
#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 [ShortRateModel](#)  
*Abstract short-rate model class.*

## 8.361 ql/ShortRateModels/onefactormodel.hpp File Reference

### 8.361.1 Detailed Description

Abstract one-factor interest rate model class.

```
#include <ql/qldefines.hpp>
```

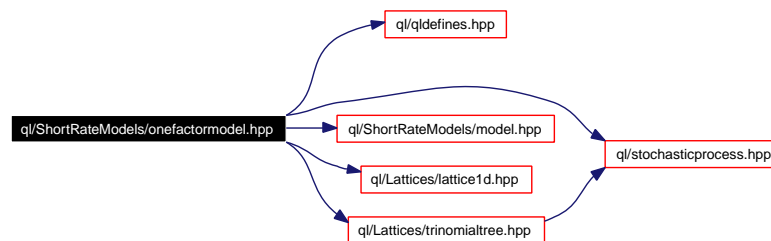
```
#include <ql/stochasticprocess.hpp>
```

```
#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.*

## 8.362 ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference

### 8.362.1 Detailed Description

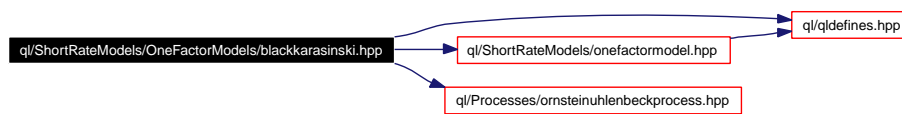
Black-Karasinski model.

```
#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.*

## 8.363 ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference

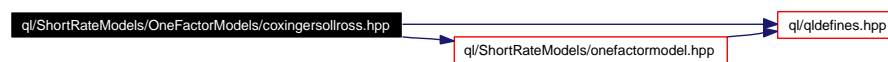
### 8.363.1 Detailed Description

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*

## 8.364 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference

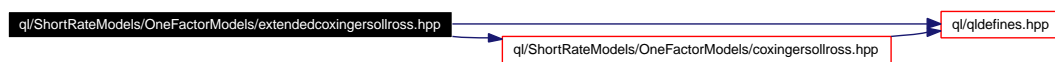
### 8.364.1 Detailed Description

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  $\varphi(t)$ .*

## 8.365 ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference

### 8.365.1 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  $\varphi(t)$ .*



## 8.366 ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference

### 8.366.1 Detailed Description

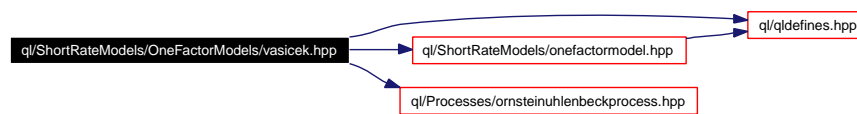
Vasicek model class.

```
#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.*

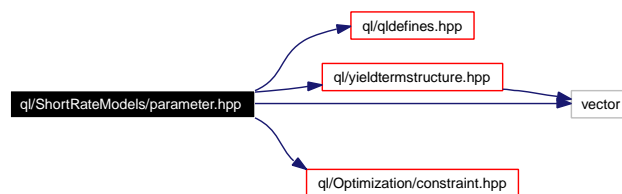
## 8.367 ql/ShortRateModels/parameter.hpp File Reference

### 8.367.1 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.*

## 8.368 ql/ShortRateModels/twofactormodel.hpp File Reference

### 8.368.1 Detailed Description

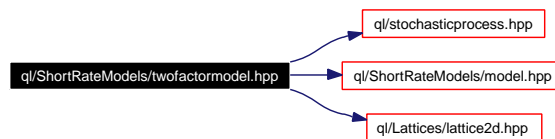
Abstract two-factor interest rate model class.

```
#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.*

## 8.369 ql/ShortRateModels/TwoFactorModels/batesmodel.hpp File Reference

### 8.369.1 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](#)

## 8.370 ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference

### 8.370.1 Detailed Description

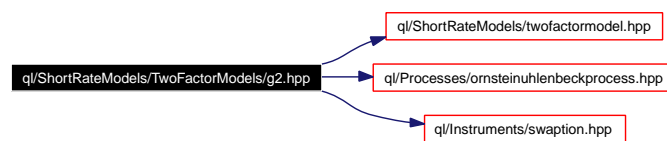
Two-factor additive Gaussian Model G2++.

```
#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  $\varphi(t)$ .*

## 8.371 ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp File Reference

### 8.371.1 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.*

## 8.372 ql/solver1d.hpp File Reference

### 8.372.1 Detailed Description

Abstract 1-D solver class.

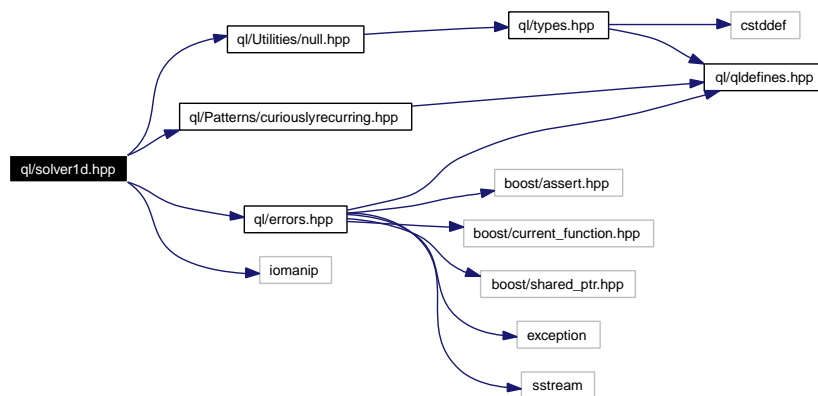
```
#include <ql/Utilities/null.hpp>
```

```
#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

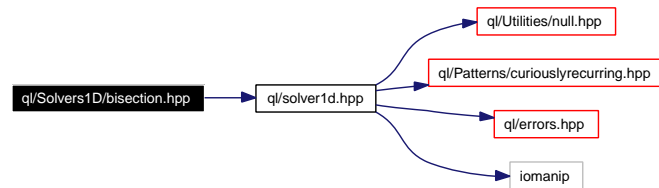
## 8.373 ql/Solvers1D/bisection.hpp File Reference

### 8.373.1 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*



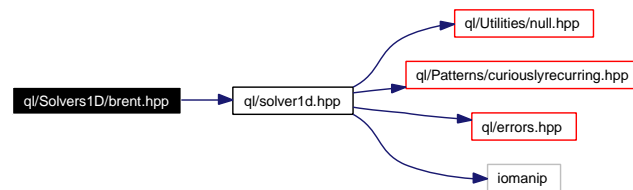
## 8.374 ql/Solvers1D/brent.hpp File Reference

### 8.374.1 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*

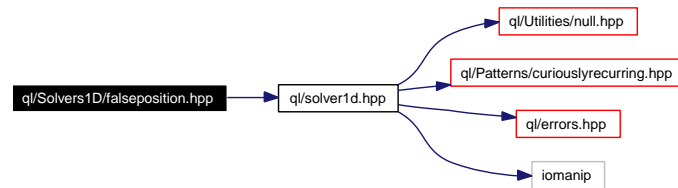
## 8.375 ql/Solvers1D/falseposition.hpp File Reference

### 8.375.1 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.*

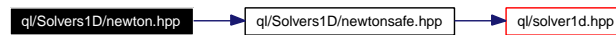
## 8.376 ql/Solvers1D/newton.hpp File Reference

### 8.376.1 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*

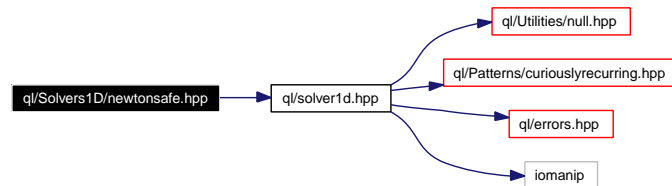
## 8.377 ql/Solvers1D/newtonsafe.hpp File Reference

### 8.377.1 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*

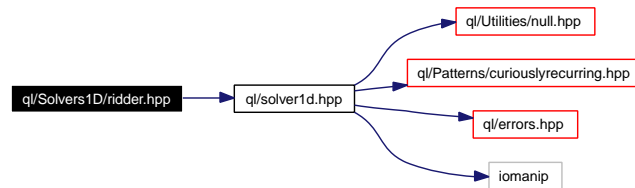
## 8.378 ql/Solvers1D/ridder.hpp File Reference

### 8.378.1 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*

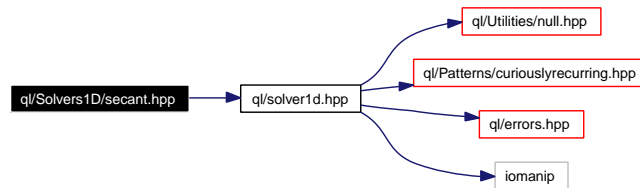
## 8.379 ql/Solvers1D/secant.hpp File Reference

### 8.379.1 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*

## 8.380 ql/stochasticprocess.hpp File Reference

### 8.380.1 Detailed Description

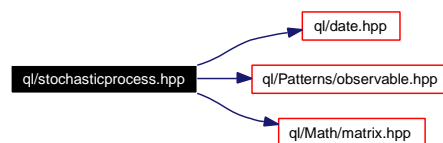
stochastic processes

```
#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*

## 8.381 ql/swaptionvolstructure.hpp File Reference

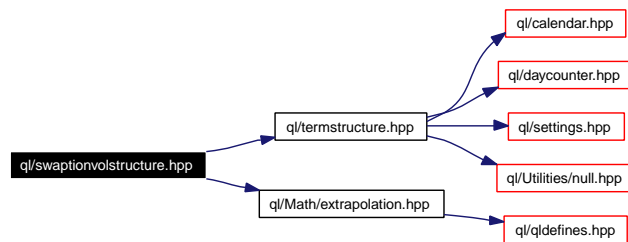
### 8.381.1 Detailed Description

Swaption volatility structure.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/Math/extrapolation.hpp>
```

Include dependency graph for swaptionvolstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwaptionVolatilityStructure](#)  
*Swaption-volatility structure*



## 8.382 ql/termstructure.hpp File Reference

### 8.382.1 Detailed Description

base class for term structures

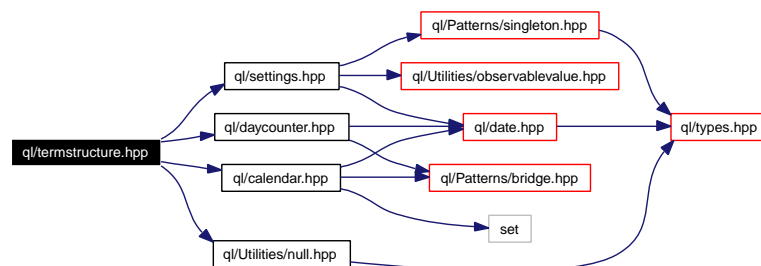
```
#include <ql/calendar.hpp>
```

```
#include <ql/daycounter.hpp>
```

```
#include <ql/settings.hpp>
```

```
#include <ql/Utilities/null.hpp>
```

Include dependency graph for termstructure.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class [TermStructure](#)  
*Basic term-structure functionality.*

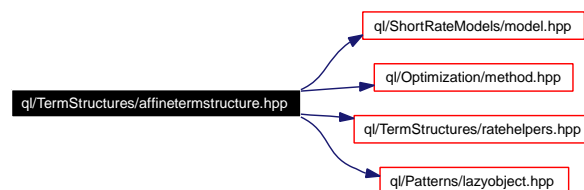
## 8.383 ql/TermStructures/affinetermstructure.hpp File Reference

### 8.383.1 Detailed Description

Affine term structure.

```
#include <ql/ShortRateModels/model.hpp>
#include <ql/Optimization/method.hpp>
#include <ql/TermStructures/ratehelpers.hpp>
#include <ql/Patterns/lazyobject.hpp>
```

Include dependency graph for affinetermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [AffineTermStructure](#)  
*Term-structure implied by an affine model.*

## 8.384 ql/TermStructures/bondhelpers.hpp File Reference

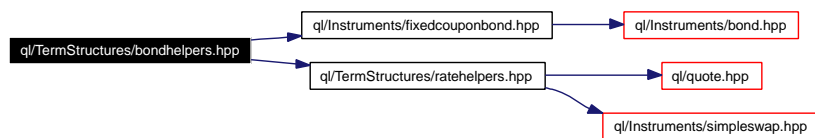
### 8.384.1 Detailed Description

bond rate helpers

```
#include <ql/Instruments/fixedcouponbond.hpp>
```

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Include dependency graph for bondhelpers.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [FixedCouponBondHelper](#)  
*fixed-coupon bond helper*

## 8.385 ql/TermStructures/bootstraptraits.hpp File Reference

### 8.385.1 Detailed Description

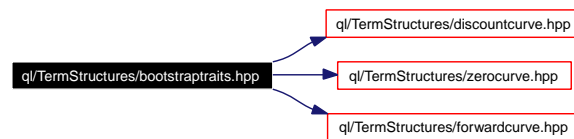
bootstrap traits

```
#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.*

## 8.386 ql/TermStructures/compoundforward.hpp File Reference

### 8.386.1 Detailed Description

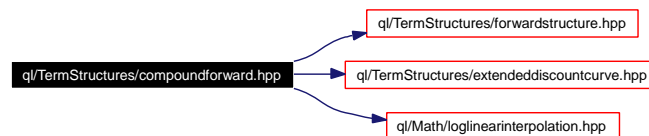
compounded forward term structure

```
#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*

## 8.387 ql/TermStructures/discountcurve.hpp File Reference

### 8.387.1 Detailed Description

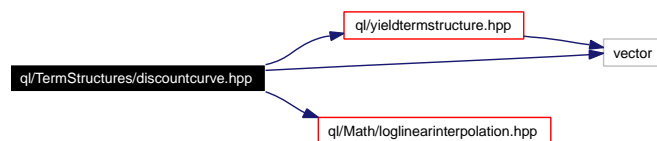
interpolated discount factor structure

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/Math/loglinearinterpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for discountcurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterpolatedDiscountCurve](#)  
*Term structure based on interpolation of discount factors.*

### Typedefs

- typedef `InterpolatedDiscountCurve< LogLinear >` [QuantLib::DiscountCurve](#)  
*Term structure based on log-linear interpolation of discount factors.*

## 8.388 ql/TermStructures/driftermstructure.hpp File Reference

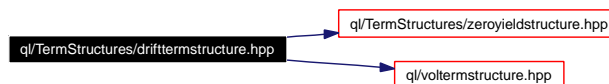
### 8.388.1 Detailed Description

Drift term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for driftermstructure.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [DriftTermStructure](#)  
*Drift term structure.*

## 8.389 ql/TermStructures/extendeddiscountcurve.hpp File Reference

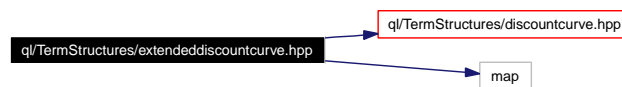
### 8.389.1 Detailed Description

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.*



## 8.390 ql/TermStructures/flatforward.hpp File Reference

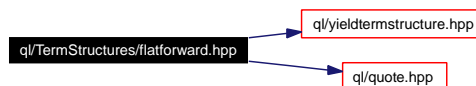
### 8.390.1 Detailed Description

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.*

## 8.391 ql/TermStructures/forwardcurve.hpp File Reference

### 8.391.1 Detailed Description

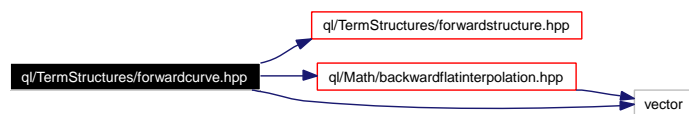
interpolated forward-rate structure

```
#include <ql/TermStructures/forwardstructure.hpp>
```

```
#include <ql/Math/backwardflatinterpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for forwardcurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterpolatedForwardCurve](#)  
*Term structure based on interpolation of forward rates.*

### Typedefs

- typedef `InterpolatedForwardCurve< BackwardFlat >` [QuantLib::ForwardCurve](#)  
*Term structure based on flat interpolation of forward rates.*

## 8.392 ql/TermStructures/forwardspreadedtermstructure.hpp File Reference

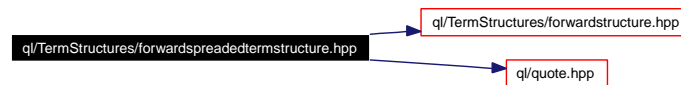
### 8.392.1 Detailed Description

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.*

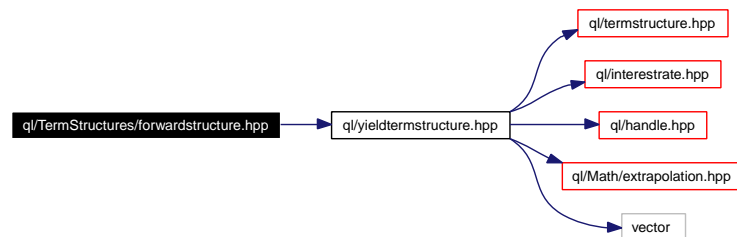
## 8.393 ql/TermStructures/forwardstructure.hpp File Reference

### 8.393.1 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.*

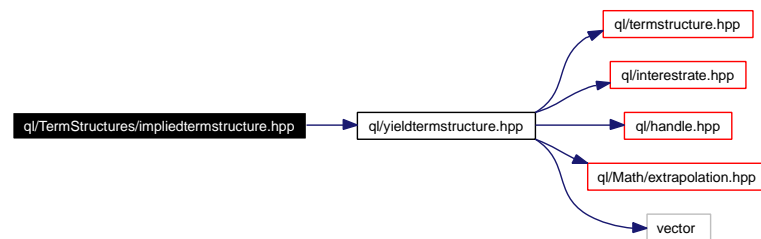
## 8.394 ql/TermStructures/IMPLIEDTERMSTRUCTURE.hpp File Reference

### 8.394.1 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.*

## 8.395 ql/TermStructures/piecewiseflatforward.hpp File Reference

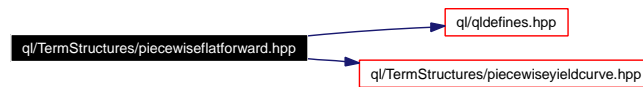
### 8.395.1 Detailed Description

piecewise flat forward term structure

```
#include <ql/qldefines.hpp>
```

```
#include <ql/TermStructures/piecewiseyieldcurve.hpp>
```

Include dependency graph for piecewiseflatforward.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `PiecewiseYieldCurve< Discount, LogLinear >` [QuantLib::PiecewiseFlatForward](#)  
*Piecewise flat-forward term structure.*

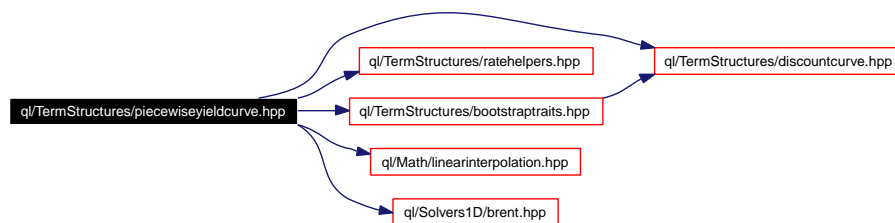
## 8.396 ql/TermStructures/piecewiseyieldcurve.hpp File Reference

### 8.396.1 Detailed Description

piecewise-interpolated term structure

```
#include <ql/TermStructures/discountcurve.hpp>
#include <ql/TermStructures/ratehelpers.hpp>
#include <ql/TermStructures/bootstraptraits.hpp>
#include <ql/Math/linearinterpolation.hpp>
#include <ql/Solvers1D/brent.hpp>
```

Include dependency graph for piecewiseyieldcurve.hpp:



### Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

### Classes

- class [PiecewiseYieldCurve](#)  
*Piecewise yield term structure.*

## 8.397 ql/TermStructures/quantotermstructure.hpp File Reference

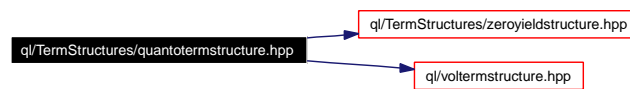
### 8.397.1 Detailed Description

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.*



## 8.398 ql/TermStructures/ratehelpers.hpp File Reference

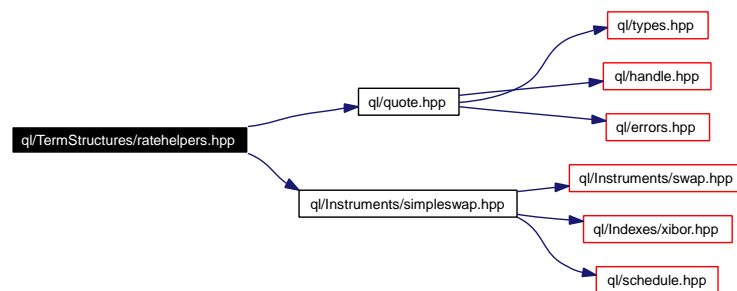
### 8.398.1 Detailed Description

rate helpers base class

```
#include <ql/quote.hpp>
```

```
#include <ql/Instruments/simpleswap.hpp>
```

Include dependency graph for ratehelpers.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [RateHelper](#)  
*Base class for rate helpers.*
- class [DepositRateHelper](#)  
*Deposit rate helper.*
- class [FraRateHelper](#)  
*Forward rate agreement helper.*
- class [FuturesRateHelper](#)  
*Interest-rate futures helper.*
- class [SwapRateHelper](#)  
*Swap rate helper*

## 8.399 ql/TermStructures/zerocurve.hpp File Reference

### 8.399.1 Detailed Description

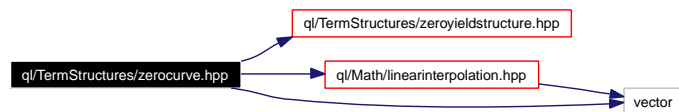
interpolated zero-rates structure

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

```
#include <ql/Math/linearinterpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for zerocurve.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [InterpolatedZeroCurve](#)  
*Term structure based on interpolation of zero yields.*

### Typedefs

- typedef `InterpolatedZeroCurve< Linear >` [QuantLib::ZeroCurve](#)  
*Term structure based on linear interpolation of zero yields.*

## 8.400 ql/TermStructures/zerospreadedtermstructure.hpp File Reference

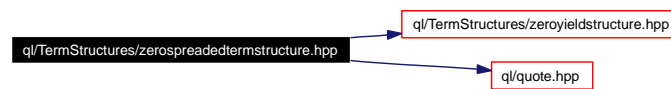
### 8.400.1 Detailed Description

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.*

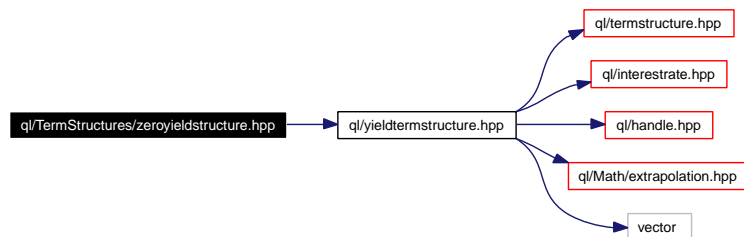
## 8.401 ql/TermStructures/zeroyieldstructure.hpp File Reference

### 8.401.1 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.*

## 8.402 ql/timegrid.hpp File Reference

### 8.402.1 Detailed Description

discrete time grid

```
#include <ql/types.hpp>
```

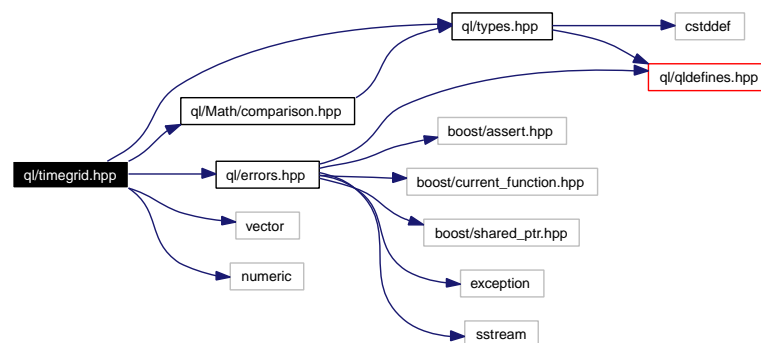
```
#include <ql/errors.hpp>
```

```
#include <ql/Math/comparison.hpp>
```

```
#include <vector>
```

```
#include <numeric>
```

Include dependency graph for timegrid.hpp:



## Namespaces

- namespace **QuantLib**

## Classes

- class **TimeGrid**  
*time grid class*

## 8.403 ql/types.hpp File Reference

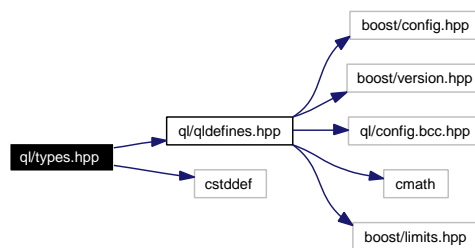
### 8.403.1 Detailed Description

Custom types.

```
#include <ql/qldefines.hpp>
```

```
#include <cstdint>
```

Include dependency graph for types.hpp:



### Namespaces

- namespace **QuantLib**

### Typedefs

- typedef `QL_INTEGER` [QuantLib::Integer](#)  
*integer number*
- typedef `QL_BIG_INTEGER` [QuantLib::BigInteger](#)  
*large integer number*
- typedef `unsigned QL_INTEGER` [QuantLib::Natural](#)  
*positive integer*
- typedef `unsigned QL_BIG_INTEGER` [QuantLib::BigNatural](#)  
*large positive integer*
- typedef `QL_REAL` [QuantLib::Real](#)  
*real number*
- typedef [Real](#) [QuantLib::Decimal](#)  
*decimal number*
- typedef `std::size_t` [QuantLib::Size](#)  
*size of a container*
- typedef [Real](#) [QuantLib::Time](#)  
*continuous quantity with 1-year units*

- typedef [Real QuantLib::DiscountFactor](#)  
*discount factor between dates*
- typedef [Real QuantLib::Rate](#)  
*interest rates*
- typedef [Real QuantLib::Spread](#)  
*spreads on interest rates*
- typedef [Real QuantLib::Volatility](#)  
*volatility*

## 8.404 ql/Utilities/dataformatters.hpp File Reference

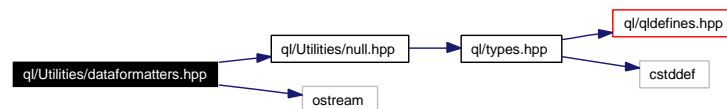
### 8.404.1 Detailed Description

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 & QuantLib::detail::operator<< (std::ostream &, const null_checker< T > &)`
- `std::ostream & QuantLib::detail::operator<< (std::ostream &, const ordinal_holder &)`
- `template<typename T> std::ostream & QuantLib::detail::operator<< (std::ostream &, const power_of_two_holder< T > &)`
- `std::ostream & QuantLib::detail::operator<< (std::ostream &, const percent_holder &)`
- `template<typename T> detail::null_checker< T > QuantLib::io::checknull (T)`  
*check for nulls before output*
- `detail::ordinal_holder QuantLib::io::ordinal (Size)`  
*outputs naturals as 1st, 2nd, 3rd...*
- `template<typename T> detail::power_of_two_holder< T > QuantLib::io::power_of_two (T)`  
*output integers as powers of two*
- `detail::percent_holder QuantLib::io::percent (Real)`  
*output reals as percentages*
- `detail::percent_holder QuantLib::io::rate (Rate)`  
*output rates and spreads as percentages*
- `detail::percent_holder QuantLib::io::volatility (Volatility)`  
*output volatilities as percentages*



## 8.405 ql/Utilities/dataparsers.hpp File Reference

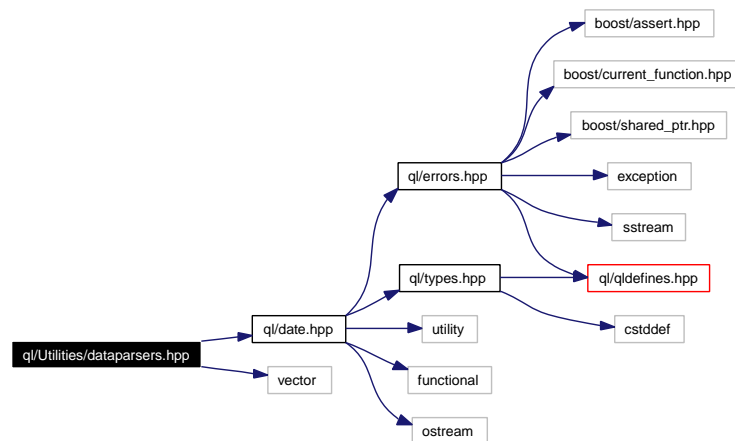
### 8.405.1 Detailed Description

Classes used to parse data for input.

```
#include <ql/date.hpp>
```

```
#include <vector>
```

Include dependency graph for dataparsers.hpp:



### Namespaces

- namespace **QuantLib**

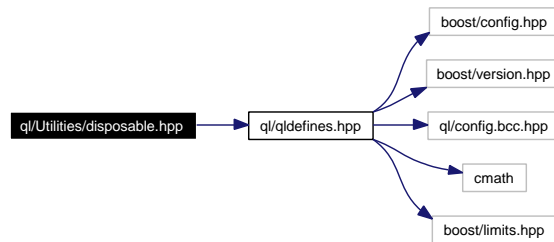
## 8.406 ql/Utilities/disposable.hpp File Reference

### 8.406.1 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*

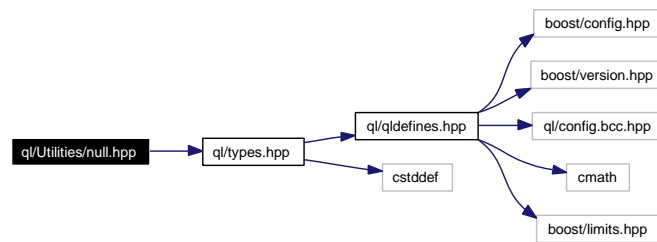
## 8.407 ql/Utilities/null.hpp File Reference

### 8.407.1 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.*

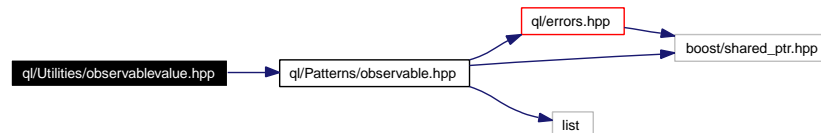
## 8.408 ql/Utilities/observablevalue.hpp File Reference

### 8.408.1 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*

## 8.409 ql/Utilities/steppingiterator.hpp File Reference

### 8.409.1 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.*

## 8.410 ql/Utilities/strings.hpp File Reference

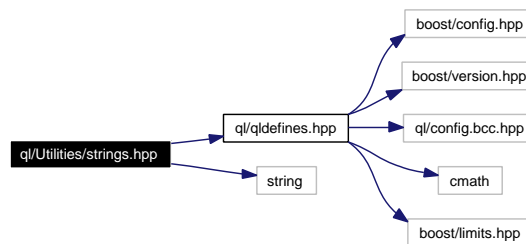
### 8.410.1 Detailed Description

string utilities

```
#include <ql/qldefines.hpp>
```

```
#include <string>
```

Include dependency graph for strings.hpp:



### Namespaces

- namespace **QuantLib**

### Functions

- `std::string QuantLib::lowercase` (`const std::string &`)
- `std::string QuantLib::uppercase` (`const std::string &`)

## 8.411 ql/Utilities/tracing.hpp File Reference

### 8.411.1 Detailed Description

tracing facilities

```
#include <ql/types.hpp>
```

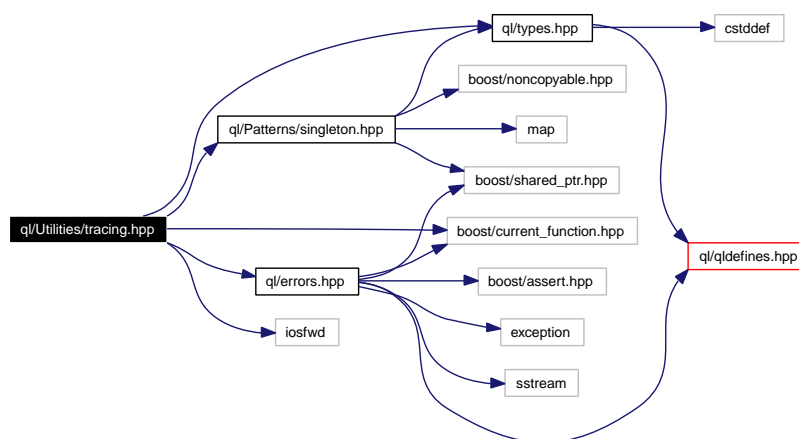
```
#include <ql/errors.hpp>
```

```
#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*



## 8.412 ql/Volatilities/blackconstantvol.hpp File Reference

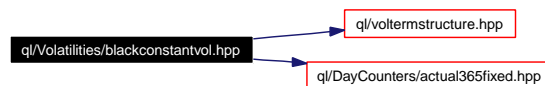
### 8.412.1 Detailed Description

Black constant volatility, no time dependence, no strike dependence.

```
#include <ql/voltermstructure.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.*

## 8.413 ql/Volatilities/blackvariancecurve.hpp File Reference

### 8.413.1 Detailed Description

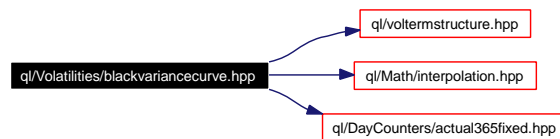
Black volatility curve modelled as variance curve.

```
#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.*

## 8.414 ql/Volatilities/blackvariancesurface.hpp File Reference

### 8.414.1 Detailed Description

Black volatility surface modelled as variance surface.

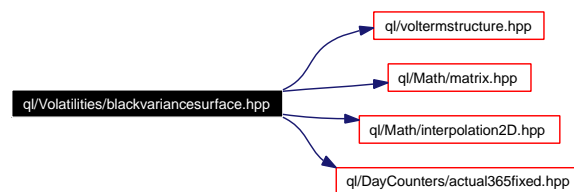
```
#include <ql/voltermstructure.hpp>
```

```
#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.*

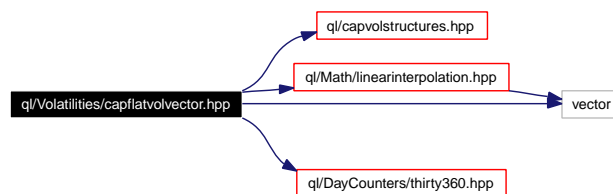
## 8.415 ql/Volatilities/capflatvolvector.hpp File Reference

### 8.415.1 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.*

## 8.416 ql/Volatilities/capletconstantvol.hpp File Reference

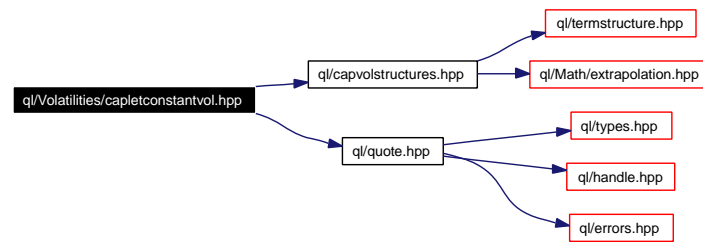
### 8.416.1 Detailed Description

Constant caplet volatility.

```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/quote.hpp>
```

Include dependency graph for capletconstantvol.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [CapletConstantVolatility](#)  
*Constant caplet volatility, no time-strike dependence.*

## 8.417 ql/Volatilities/capletvariancecurve.hpp File Reference

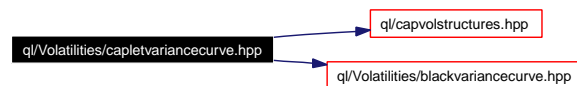
### 8.417.1 Detailed Description

caplet variance curve

```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for capletvariancecurve.hpp:



### Namespaces

- namespace **QuantLib**

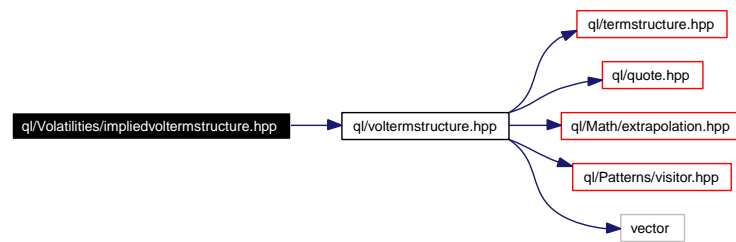
## 8.418 ql/Volatilities/impliedvoltermstructure.hpp File Reference

### 8.418.1 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.*

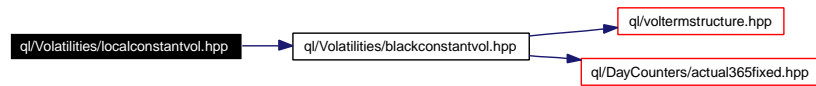
## 8.419 ql/Volatilities/localconstantvol.hpp File Reference

### 8.419.1 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.*



## 8.420 ql/Volatilities/localvolcurve.hpp File Reference

### 8.420.1 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.*

## 8.421 ql/Volatilities/localvolsurface.hpp File Reference

### 8.421.1 Detailed Description

Local volatility surface derived from a Black vol surface.

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

Include dependency graph for localvolsurface.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [LocalVolSurface](#)  
*Local volatility surface derived from a Black vol surface.*

## 8.422 ql/Volatilities/swaptionvolmatrix.hpp File Reference

### 8.422.1 Detailed Description

Swaption at-the-money volatility matrix.

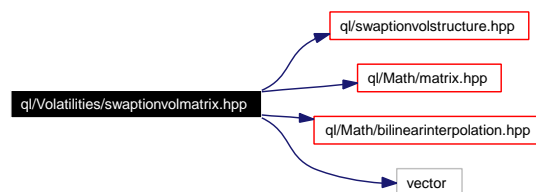
```
#include <ql/swaptionvolstructure.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Math/bilinearinterpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for swaptionvolmatrix.hpp:



### Namespaces

- namespace **QuantLib**

### Classes

- class [SwaptionVolatilityMatrix](#)  
*At-the-money swaption-volatility matrix.*

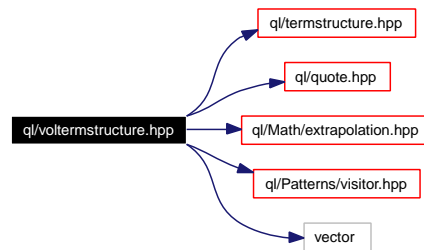
## 8.423 ql/voltermstructure.hpp File Reference

### 8.423.1 Detailed Description

Volatility term structures.

```
#include <ql/termstructure.hpp>
#include <ql/quote.hpp>
#include <ql/Math/extrapolation.hpp>
#include <ql/Patterns/visitor.hpp>
#include <vector>
```

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.*

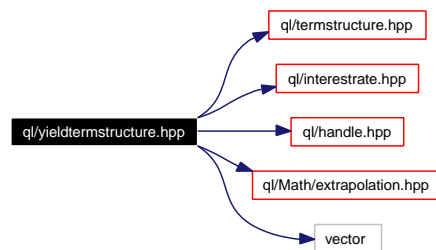
## 8.424 ql/yieldtermstructure.hpp File Reference

### 8.424.1 Detailed Description

Interest-rate term structure.

```
#include <ql/termstructure.hpp>
#include <ql/interestrate.hpp>
#include <ql/handle.hpp>
#include <ql/Math/extrapolation.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 #include <ql/quantlib.hpp>
23 #include <boost/timer.hpp>
24 #include <iostream>
25 #include <iomanip>
26
27 using namespace QuantLib;
28
29 #if defined(QL_ENABLE_SESSIONS)
30 namespace QuantLib {
31
32     Integer sessionId() { return 0; }
33
34 }
35 #endif
36
37
38 //Number of swaptions to be calibrated to...
39
40 Size numRows = 5;
41 Size numCols = 5;
42
43 Integer swapLengths[] = {
44     1, 2, 3, 4, 5};
45 Volatility swaptionVols[] = {
46     0.1490, 0.1340, 0.1228, 0.1189, 0.1148,
47     0.1290, 0.1201, 0.1146, 0.1108, 0.1040,
48     0.1149, 0.1112, 0.1070, 0.1010, 0.0957,
49     0.1047, 0.1021, 0.0980, 0.0951, 0.1270,
50     0.1000, 0.0950, 0.0900, 0.1230, 0.1160};
51
52 void calibrateModel(const boost::shared_ptr<ShortRateModel>& model,
53                     const std::vector<boost::shared_ptr<CalibrationHelper> >&
54                     helpers,
55                     Real lambda) {
56
57     LevenbergMarquardt om;
58     model->calibrate(helpers, om);
59
60     // Output the implied Black volatilities
```

```

61     for (Size i=0; i<numRows; i++) {
62         Size j = numCols - i - 1; // 1x5, 2x4, 3x3, 4x2, 5x1
63         Size k = i*numCols + j;
64         Real npv = helpers[i]->modelValue();
65         Volatility implied = helpers[i]->impliedVolatility(npv, 1e-4,
66                                                         1000, 0.05, 0.50);
67         Volatility diff = implied - swaptionVols[k];
68
69         std::cout << i+1 << "x" << swapLengths[j]
70                 << std::setprecision(5) << std::noshowpos
71                 << ": model " << std::setw(7) << io::volatility(implied)
72                 << ", market " << std::setw(7)
73                 << io::volatility(swaptionVols[k])
74                 << " (" << std::setw(7) << std::showpos
75                 << io::volatility(diff) << std::noshowpos << ")\n";
76     }
77 }
78
79 int main(int, char* [])
80 {
81     try {
82         QL_IO_INIT
83
84         boost::timer timer;
85         std::cout << std::endl;
86
87         Date todaysDate(15, February, 2002);
88         Calendar calendar = TARGET();
89         Date settlementDate(19, February, 2002);
90         Settings::instance().evaluationDate() = todaysDate;
91
92         // flat yield term structure impling 1x5 swap at 5%
93         boost::shared_ptr<Quote> flatRate(new SimpleQuote(0.04875825));
94         boost::shared_ptr<FlatForward> myTermStructure(
95             new FlatForward(settlementDate, Handle<Quote>(flatRate),
96                             Actual365Fixed()));
97         Handle<YieldTermStructure> rhTermStructure;
98         rhTermStructure.linkTo(myTermStructure);
99
100        // Define the ATM/OTM/ITM swaps
101        Frequency fixedLegFrequency = Annual;
102        BusinessDayConvention fixedLegConvention = Unadjusted;
103        BusinessDayConvention floatingLegConvention = ModifiedFollowing;
104        DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
105        Frequency floatingLegFrequency = Semiannual;
106        bool payFixedRate = true;
107        Integer fixingDays = 2;
108        Rate dummyFixedRate = 0.03;
109        boost::shared_ptr<Xibor> indexSixMonths(new
110            Euribor(6, Months, rhTermStructure));
111
112        Date startDate = calendar.advance(settlementDate, 1, Years,
113                                         floatingLegConvention);
114        Date maturity = calendar.advance(startDate, 5, Years,
115                                         floatingLegConvention);
116        Schedule fixedSchedule(calendar, startDate, maturity,
117                               fixedLegFrequency, fixedLegConvention);
118        Schedule floatSchedule(calendar, startDate, maturity,
119                               floatingLegFrequency, floatingLegConvention);
120        boost::shared_ptr<VanillaSwap> swap(new VanillaSwap(
121            payFixedRate, 1000.0,
122            fixedSchedule, dummyFixedRate, fixedLegDayCounter,
123            floatSchedule, indexSixMonths, fixingDays, 0.0,
124            indexSixMonths->dayCounter(), rhTermStructure));
125        Rate fixedATMRate = swap->fairRate();
126        Rate fixedOTMRate = fixedATMRate * 1.2;
127        Rate fixedITMRate = fixedATMRate * 0.8;

```



```

128
129     boost::shared_ptr<VanillaSwap> atmSwap(new VanillaSwap(
130         payFixedRate, 1000.0,
131         fixedSchedule, fixedATMRate, fixedLegDayCounter,
132         floatSchedule, indexSixMonths, fixingDays, 0.0,
133         indexSixMonths->dayCounter(), rhTermStructure));
134     boost::shared_ptr<VanillaSwap> otmSwap(new VanillaSwap(
135         payFixedRate, 1000.0,
136         fixedSchedule, fixedOTMRate, fixedLegDayCounter,
137         floatSchedule, indexSixMonths, fixingDays, 0.0,
138         indexSixMonths->dayCounter(), rhTermStructure));
139     boost::shared_ptr<VanillaSwap> itmSwap(new VanillaSwap(
140         payFixedRate, 1000.0,
141         fixedSchedule, fixedITMRate, fixedLegDayCounter,
142         floatSchedule, indexSixMonths, fixingDays, 0.0,
143         indexSixMonths->dayCounter(), rhTermStructure));
144
145     // defining the swaptions to be used in model calibration
146     std::vector<Period> swaptionMaturities;
147     swaptionMaturities.push_back(Period(1, Years));
148     swaptionMaturities.push_back(Period(2, Years));
149     swaptionMaturities.push_back(Period(3, Years));
150     swaptionMaturities.push_back(Period(4, Years));
151     swaptionMaturities.push_back(Period(5, Years));
152
153     std::vector<boost::shared_ptr<CalibrationHelper> > swaptions;
154
155     // List of times that have to be included in the timegrid
156     std::list<Time> times;
157
158     Size i;
159     for (i=0; i<numRows; i++) {
160         Size j = numCols - i -1; // 1x5, 2x4, 3x3, 4x2, 5x1
161         Size k = i*numCols + j;
162         boost::shared_ptr<Quote> vol(new SimpleQuote(swaptionVols[k]));
163         swaptions.push_back(boost::shared_ptr<CalibrationHelper>(new
164             SwaptionHelper(swaptionMaturities[i],
165                 Period(swapLengths[j], Years),
166                 Handle<Quote>(vol),
167                 indexSixMonths,
168                 indexSixMonths->frequency(),
169                 indexSixMonths->dayCounter(),
170                 indexSixMonths->dayCounter(),
171                 rhTermStructure)));
172         swaptions.back()->addTimesTo(times);
173     }
174
175     // Building time-grid
176     TimeGrid grid(times.begin(), times.end(), 30);
177
178
179     // defining the models
180     boost::shared_ptr<G2> modelG2(new G2(rhTermStructure));
181     boost::shared_ptr<HullWhite> modelHW(new HullWhite(rhTermStructure));
182     boost::shared_ptr<HullWhite> modelHW2(new HullWhite(rhTermStructure));
183     boost::shared_ptr<BlackKarasinski> modelBK(
184         new BlackKarasinski(rhTermStructure));
185
186
187     // model calibrations
188
189     std::cout << "G2 (analytic formulae) calibration" << std::endl;
190     for (i=0; i<swaptions.size(); i++)
191         swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
192             new G2SwaptionEngine(modelG2, 6.0, 16)));
193
194     calibrateModel(modelG2, swaptions, 0.05);

```

```

195     std::cout << "calibrated to:\n"
196         << "a      = " << modelG2->params()[0] << ", "
197         << "sigma = " << modelG2->params()[1] << "\n"
198         << "b      = " << modelG2->params()[2] << ", "
199         << "eta    = " << modelG2->params()[3] << "\n"
200         << "rho    = " << modelG2->params()[4]
201         << std::endl << std::endl;
202
203
204
205     std::cout << "Hull-White (analytic formulae) calibration" << std::endl;
206     for (i=0; i<swaptions.size(); i++)
207         swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
208             new JamshidianSwaptionEngine(modelHW)));
209
210     calibrateModel(modelHW, swaptions, 0.05);
211     std::cout << "calibrated to:\n"
212         << "a = " << modelHW->params()[0] << ", "
213         << "sigma = " << modelHW->params()[1]
214         << std::endl << std::endl;
215
216     std::cout << "Hull-White (numerical) calibration" << std::endl;
217     for (i=0; i<swaptions.size(); i++)
218         swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
219             new TreeSwaptionEngine(modelHW2,grid)));
220
221     calibrateModel(modelHW2, swaptions, 0.05);
222     std::cout << "calibrated to:\n"
223         << "a = " << modelHW2->params()[0] << ", "
224         << "sigma = " << modelHW2->params()[1]
225         << std::endl << std::endl;
226
227     std::cout << "Black-Karasinski (numerical) calibration" << std::endl;
228     for (i=0; i<swaptions.size(); i++)
229         swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
230             new TreeSwaptionEngine(modelBK,grid)));
231
232     calibrateModel(modelBK, swaptions, 0.05);
233     std::cout << "calibrated to:\n"
234         << "a = " << modelBK->params()[0] << ", "
235         << "sigma = " << modelBK->params()[1]
236         << std::endl << std::endl;
237
238
239     // ATM Bermudan swaption pricing
240
241     std::cout << "Payer bermudan swaption "
242         << "struck at " << io::rate(fixedATMRate)
243         << " (ATM)" << std::endl;
244
245     std::vector<Date> bermudanDates;
246     const std::vector<boost::shared_ptr<CashFlow> >& leg =
247         swap->fixedLeg();
248     for (i=0; i<leg.size(); i++) {
249         boost::shared_ptr<Coupon> coupon =
250             boost::dynamic_pointer_cast<Coupon>(leg[i]);
251         bermudanDates.push_back(coupon->accrualStartDate());
252     }
253
254     boost::shared_ptr<Exercise> bermudanExercise(
255         new BermudanExercise(bermudanDates));
256
257     Swaption bermudanSwaption(atmSwap, bermudanExercise, rhTermStructure,
258         boost::shared_ptr<PricingEngine>());
259
260     // Do the pricing for each model
261

```

```

262 // G2 price the European swaption here, it should switch to bermudan
263 bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
264     TreeSwaptionEngine(modelG2, 50)));
265 std::cout << "G2:      " << bermudanSwaption.NPV() << std::endl;
266
267 bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
268     new TreeSwaptionEngine(modelHW, 50)));
269 std::cout << "HW:      " << bermudanSwaption.NPV() << std::endl;
270
271 bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
272     TreeSwaptionEngine(modelHW2, 50)));
273 std::cout << "HW (num): " << bermudanSwaption.NPV() << std::endl;
274
275 bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
276     TreeSwaptionEngine(modelBK, 50)));
277 std::cout << "BK:      " << bermudanSwaption.NPV() << std::endl;
278
279
280 // OTM Bermudan swaption pricing
281
282 std::cout << "Payer bermudan swaption "
283     << "struck at " << io::rate(fixedOTMRate)
284     << " (OTM)" << std::endl;
285
286 Swaption otmBermudanSwaption(otmSwap,bermudanExercise,rhTermStructure,
287     boost::shared_ptr<PricingEngine>());
288
289 // Do the pricing for each model
290 otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
291     new TreeSwaptionEngine(modelG2, 50)));
292 std::cout << "G2:      " << otmBermudanSwaption.NPV() << std::endl;
293
294 otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
295     new TreeSwaptionEngine(modelHW, 50)));
296 std::cout << "HW:      " << otmBermudanSwaption.NPV() << std::endl;
297
298 otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
299     new TreeSwaptionEngine(modelHW2, 50)));
300 std::cout << "HW (num): " << otmBermudanSwaption.NPV() << std::endl;
301
302 otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
303     new TreeSwaptionEngine(modelBK, 50)));
304 std::cout << "BK:      " << otmBermudanSwaption.NPV() << std::endl;
305
306
307 // ITM Bermudan swaption pricing
308
309 std::cout << "Payer bermudan swaption "
310     << "struck at " << io::rate(fixedITMRate)
311     << " (ITM)" << std::endl;
312
313 Swaption itmBermudanSwaption(itmSwap,bermudanExercise,rhTermStructure,
314     boost::shared_ptr<PricingEngine>());
315
316 // Do the pricing for each model
317 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
318     new TreeSwaptionEngine(modelG2, 50)));
319 std::cout << "G2:      " << itmBermudanSwaption.NPV() << std::endl;
320
321 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
322     new TreeSwaptionEngine(modelHW, 50)));
323 std::cout << "HW:      " << itmBermudanSwaption.NPV() << std::endl;
324
325 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
326     new TreeSwaptionEngine(modelHW2, 50)));
327 std::cout << "HW (num): " << itmBermudanSwaption.NPV() << std::endl;
328

```

```
329     itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
330         new TreeSwaptionEngine(modelBK, 50)));
331     std::cout << "BK:      " << itmBermudanSwaption.NPV() << std::endl;
332
333     Real seconds = timer.elapsed();
334     Integer hours = int(seconds/3600);
335     seconds -= hours * 3600;
336     Integer minutes = int(seconds/60);
337     seconds -= minutes * 60;
338     std::cout << " \nRun completed in ";
339     if (hours > 0)
340         std::cout << hours << " h ";
341     if (hours > 0 || minutes > 0)
342         std::cout << minutes << " m ";
343     std::cout << std::fixed << std::setprecision(0)
344         << seconds << " s\n" << std::endl;
345
346     return 0;
347 } catch (std::exception& e) {
348     std::cout << e.what() << std::endl;
349     return 1;
350 } catch (...) {
351     std::cout << "unknown error" << std::endl;
352     return 1;
353 }
354 }
355
```

## 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 #include <ql/quantlib.hpp>
22 #include <boost/timer.hpp>
23 #include <iostream>
24 #include <iomanip>
25
26 #define LENGTH(a) (sizeof(a)/sizeof(a[0]))
27
28 using namespace QuantLib;
29
30 #if defined(QL_ENABLE_SESSIONS)
31 namespace QuantLib {
32
33     Integer sessionId() { return 0; }
34
35 }
36 #endif
37
38
39 int main(int argc, char* argv[])
40 {
41     try {
42
43         QL_IO_INIT
44
45         boost::timer timer;
46         std::cout << std::endl;
47
48         Option::Type type(Option::Put);
49         Real underlying = 36.0;
50         Real spreadRate = 0.005;
51
52         Spread dividendYield = 0.02;
53         Rate riskFreeRate = 0.06;
54         Volatility volatility = 0.20;
55
56         Integer settlementDays = 3;
57         Integer length = 5;
58         Real redemption = 100.0;
59         Real conversionRatio = redemption/underlying; // at the money
60
61         // set up dates/schedules
62         Calendar calendar = TARGET();
63         Date today = calendar.adjust(Date::todaysDate());
64
65         Settings::instance().evaluationDate() = today;
66         Date settlementDate = calendar.advance(today, settlementDays, Days);
67         Date exerciseDate = calendar.advance(settlementDate, length, Years);
68         Date issueDate = calendar.advance(exerciseDate, -length, Years);
69
70         BusinessDayConvention convention = ModifiedFollowing;
71
72         Frequency frequency = Annual;
73
74         Schedule schedule(calendar,issueDate,exerciseDate,
75                           frequency, convention,Date(), true);
76
77         DividendSchedule dividends;
78         CallabilitySchedule callability;
79
80         std::vector<Real> coupons(1, 0.05);

```

```

81
82     DayCounter bondDayCount = Thirty360();
83
84     Integer callLength[] = { 2, 4 }; // Call dates, years 2, 4.
85     Integer putLength[] = { 3 }; // Put dates year 3
86
87     Real callPrices[] = { 101.5, 100.85 };
88     Real putPrices[] = { 105.0 };
89
90     // Load call schedules
91     for (Size i=0; i<LENGTH(callLength); i++) {
92         callability.push_back(
93             Callability(Price(callPrices[i], Price::Clean),
94                         Callability::Call,
95                         schedule.date(callLength[i])));
96     }
97
98     for (Size j=0; j<LENGTH(putLength); j++) {
99         callability.push_back(
100             Callability(Price(putPrices[j], Price::Clean),
101                         Callability::Put,
102                         schedule.date(putLength[j])));
103     }
104
105     // Assume dividends are paid every 6 months.
106     /* for (Date d = today + 6*Months;
107         d < exerciseDate;
108         d += 6*Months) {
109         dividends.push_back(
110             boost::shared_ptr<CashFlow>(new FixedDividend(1.0, d)));
111     }*/
112
113     DayCounter dayCounter = Actual365Fixed();
114     Time maturity = dayCounter.yearFraction(settlementDate,
115                                             exerciseDate);
116
117     std::cout << "option type = " << type << std::endl;
118     std::cout << "Time to maturity = " << maturity
119               << std::endl;
120     std::cout << "Underlying price = " << underlying
121               << std::endl;
122     std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
123               << std::endl;
124     std::cout << "Dividend yield = " << io::rate(dividendYield)
125               << std::endl;
126     std::cout << "Volatility = " << io::volatility(volatility)
127               << std::endl;
128     std::cout << std::endl;
129
130     std::string method;
131     std::cout << std::endl ;
132
133     // write column headings
134     Size widths[] = { 35, 14, 14 };
135     Size totalWidth = widths[0] + widths[1] + widths[2];
136     std::string rule(totalWidth, '-'), dblrule(totalWidth, '=');
137
138     std::cout << dblrule << std::endl;
139     std::cout << "Tsiveriotis-Fernandes method" << std::endl;
140     std::cout << dblrule << std::endl;
141     std::cout << std::setw(widths[0]) << std::left << "Tree type"
142               << std::setw(widths[1]) << std::left << "European"
143               << std::setw(widths[1]) << std::left << "American"
144               << std::endl;
145     std::cout << rule << std::endl;
146
147     boost::shared_ptr<Exercise> exercise(

```

```

148         new EuropeanExercise(exerciseDate));
149     boost::shared_ptr<Exercise> amExercise(
150         new AmericanExercise(settlementDate,
151             exerciseDate));
152
153     Handle<Quote> underlyingH(
154         boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
155
156     Handle<YieldTermStructure> flatTermStructure(
157         boost::shared_ptr<YieldTermStructure>(
158             new FlatForward(settlementDate, riskFreeRate, dayCounter)));
159
160     Handle<YieldTermStructure> flatDividendTS(
161         boost::shared_ptr<YieldTermStructure>(
162             new FlatForward(settlementDate, dividendYield, dayCounter)));
163
164     Handle<BlackVolTermStructure> flatVolTS(
165         boost::shared_ptr<BlackVolTermStructure>(
166             new BlackConstantVol(settlementDate, volatility, dayCounter)));
167
168
169     boost::shared_ptr<BlackScholesProcess> stochasticProcess(
170         new BlackScholesProcess(underlyingH,
171             flatDividendTS,
172             flatTermStructure,
173             flatVolTS));
174
175     Size timeSteps = 801;
176
177     Handle<Quote> creditSpread(
178         boost::shared_ptr<Quote>(new SimpleQuote(spreadRate)));
179
180     boost::shared_ptr<Quote> rate(new SimpleQuote(riskFreeRate));
181
182     Handle<YieldTermStructure> discountCurve(
183         boost::shared_ptr<YieldTermStructure>(
184             new FlatForward(today, Handle<Quote>(rate), dayCounter)));
185
186     boost::shared_ptr<PricingEngine> engine(
187         new BinomialConvertibleEngine<JarrowRudd>(timeSteps));
188
189     ConvertibleFixedCouponBond europeanBond(
190         stochasticProcess, exercise, engine,
191         conversionRatio, dividends, callability,
192         creditSpread, issueDate, settlementDays,
193         coupons, bondDayCount, schedule, redemption);
194
195     ConvertibleFixedCouponBond americanBond(
196         stochasticProcess, amExercise, engine,
197         conversionRatio, dividends, callability,
198         creditSpread, issueDate, settlementDays,
199         coupons, bondDayCount, schedule, redemption);
200
201     method = "Jarrow-Rudd";
202     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
203         new BinomialConvertibleEngine<JarrowRudd>(timeSteps)));
204     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
205         new BinomialConvertibleEngine<JarrowRudd>(timeSteps)));
206     std::cout << std::setw(widths[0]) << std::left << method
207         << std::fixed
208         << std::setw(widths[1]) << std::left << europeanBond.NPV()
209         << std::setw(widths[2]) << std::left << americanBond.NPV()
210         << std::endl;
211
212     method = "Cox-Ross-Rubinstein";
213     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
214         new BinomialConvertibleEngine<CoxRossRubinstein>(timeSteps)));

```

```

215     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
216         new BinomialConvertibleEngine<CoxRossRubinstein>(timeSteps)));
217     std::cout << std::setw(widths[0]) << std::left << method
218         << std::fixed
219         << std::setw(widths[1]) << std::left << europeanBond.NPV()
220         << std::setw(widths[2]) << std::left << americanBond.NPV()
221         << std::endl;
222
223     method = "Additive equiprobabilities";
224     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
225         new BinomialConvertibleEngine<AdditiveEQPBinoomialTree>(timeSteps)));
226     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
227         new BinomialConvertibleEngine<AdditiveEQPBinoomialTree>(timeSteps)));
228     std::cout << std::setw(widths[0]) << std::left << method
229         << std::fixed
230         << std::setw(widths[1]) << std::left << europeanBond.NPV()
231         << std::setw(widths[2]) << std::left << americanBond.NPV()
232         << std::endl;
233
234     method = "Trigeorgis";
235     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
236         new BinomialConvertibleEngine<Trigeorgis>(timeSteps)));
237     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
238         new BinomialConvertibleEngine<Trigeorgis>(timeSteps)));
239     std::cout << std::setw(widths[0]) << std::left << method
240         << std::fixed
241         << std::setw(widths[1]) << std::left << europeanBond.NPV()
242         << std::setw(widths[2]) << std::left << americanBond.NPV()
243         << std::endl;
244
245     method = "Tian";
246     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
247         new BinomialConvertibleEngine<Tian>(timeSteps)));
248     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
249         new BinomialConvertibleEngine<Tian>(timeSteps)));
250     std::cout << std::setw(widths[0]) << std::left << method
251         << std::fixed
252         << std::setw(widths[1]) << std::left << europeanBond.NPV()
253         << std::setw(widths[2]) << std::left << americanBond.NPV()
254         << std::endl;
255
256     method = "Leisen-Reimer";
257     europeanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
258         new BinomialConvertibleEngine<LeisenReimer>(timeSteps)));
259     americanBond.setPricingEngine(boost::shared_ptr<PricingEngine>(
260         new BinomialConvertibleEngine<LeisenReimer>(timeSteps)));
261     std::cout << std::setw(widths[0]) << std::left << method
262         << std::fixed
263         << std::setw(widths[1]) << std::left << europeanBond.NPV()
264         << std::setw(widths[2]) << std::left << americanBond.NPV()
265         << std::endl;
266
267     std::cout << dblrule << std::endl;
268
269     Real seconds = timer.elapsed();
270     Integer hours = int(seconds/3600);
271     seconds -= hours * 3600;
272     Integer minutes = int(seconds/60);
273     seconds -= minutes * 60;
274     std::cout << " \nRun completed in ";
275     if (hours > 0)
276         std::cout << hours << " h ";
277     if (hours > 0 || minutes > 0)
278         std::cout << minutes << " m ";
279     std::cout << std::fixed << std::setprecision(0)
280         << seconds << " s\n" << std::endl;
281

```



```
282         return 0;
283     } catch (std::exception& e) {
284         std::cout << e.what() << std::endl;
285         return 1;
286     } catch (...) {
287         std::cout << "unknown error" << std::endl;
288         return 1;
289     }
290 }
291 }
292
```



```

80     // value of the option
81     DiscountFactor rDiscount = std::exp(-r_*maturity_);
82     DiscountFactor qDiscount = 1.0;
83     Real forward = s0_*qDiscount/rDiscount;
84     Real variance = sigma_*sigma_*maturity_;
85     boost::shared_ptr<StrikedTypePayoff> payoff(
86         new PlainVanillaPayoff(payoff_));
87     BlackFormula black(forward,rDiscount,variance,payoff);
88     std::cout << "Option value: " << black.value() << std::endl;
89
90     // store option's vega, since Derman and Kamal's formula needs it
91     vega_ = black.vega(maturity_);
92
93     std::cout << std::endl;
94     std::cout <<
95         "          |          | P&L  \t|  P&L      | Derman&Kamal | P&L"
96         "          \t| P&L" << std::endl;
97
98     std::cout <<
99         "samples | trades | Mean \t| Std Dev | Formula      |"
100        " skewness \t| kurt." << std::endl;
101
102     std::cout << "-----"
103         "-----" << std::endl;
104 }
105
106 // the actual replication error computation
107 void compute(Size nTimeSteps, Size nSamples);
108 private:
109     Time maturity_;
110     PlainVanillaPayoff payoff_;
111     Real s0_;
112     Volatility sigma_;
113     Rate r_;
114     Real vega_;
115 };
116
117 // The key for the MonteCarlo simulation is to have a PathPricer that
118 // implements a value(const Path& path) method.
119 // This method prices the portfolio for each Path of the random variable
120 class ReplicationPathPricer : public PathPricer<Path> {
121 public:
122     // real constructor
123     ReplicationPathPricer(Option::Type type,
124                          Real strike,
125                          Rate r,
126                          Time maturity,
127                          Volatility sigma)
128     : type_(type), strike_(strike),
129       r_(r), maturity_(maturity), sigma_(sigma) {
130         QL_REQUIRE(strike_ > 0.0, "strike must be positive");
131         QL_REQUIRE(r_ >= 0.0,
132                    "risk free rate (r) must be positive or zero");
133         QL_REQUIRE(maturity_ > 0.0, "maturity must be positive");
134         QL_REQUIRE(sigma_ >= 0.0,
135                    "volatility (sigma) must be positive or zero");
136     }
137
138     // The value() method encapsulates the pricing code
139     Real operator()(const Path& path) const;
140
141 private:
142     Option::Type type_;
143     Real strike_;
144     Rate r_;
145     Time maturity_;
146     Volatility sigma_;

```

```

147 };
148
149
150 // Compute Replication Error as in the Derman and Kamal's research note
151 int main(int, char* [])
152 {
153     try {
154         QL_IO_INIT
155
156         boost::timer timer;
157         std::cout << std::endl;
158
159         Time maturity = 1.0/12.0;    // 1 month
160         Real strike = 100;
161         Real underlying = 100;
162         Volatility volatility = 0.20; // 20%
163         Rate riskFreeRate = 0.05; // 5%
164         ReplicationError rp(Option::Call, maturity, strike, underlying,
165                             volatility, riskFreeRate);
166
167         Size scenarios = 50000;
168         Size hedgesNum;
169
170         hedgesNum = 21;
171         rp.compute(hedgesNum, scenarios);
172
173         hedgesNum = 84;
174         rp.compute(hedgesNum, scenarios);
175
176         Real seconds = timer.elapsed();
177         Integer hours = int(seconds/3600);
178         seconds -= hours * 3600;
179         Integer minutes = int(seconds/60);
180         seconds -= minutes * 60;
181         std::cout << " \nRun completed in ";
182         if (hours > 0)
183             std::cout << hours << " h ";
184         if (hours > 0 || minutes > 0)
185             std::cout << minutes << " m ";
186         std::cout << std::fixed << std::setprecision(0)
187             << seconds << " s\n" << std::endl;
188
189         return 0;
190     } catch (std::exception& e) {
191         std::cout << e.what() << std::endl;
192         return 1;
193     } catch (...) {
194         std::cout << "unknown error" << std::endl;
195         return 1;
196     }
197 }
198
199
200 /* The actual computation of the Profit&Loss for each single path.
201
202 In each scenario N reheding trades spaced evenly in time over
203 the life of the option are carried out, using the Black-Scholes
204 hedge ratio.
205 */
206 Real ReplicationPathPricer::operator()(const Path& path) const {
207
208     Size n = path.length()-1;
209     QL_REQUIRE(n>0, "the path cannot be empty");
210
211     // discrete hedging interval
212     Time dt = maturity_/n;
213

```

```

214 // For simplicity, we assume the stock pays no dividends.
215 Rate stockDividendYield = 0.0;
216
217 // let's start
218 Time t = 0;
219
220 // stock value at t=0
221 Real stock = path.front();
222
223 // money account at t=0
224 Real money_account = 0.0;
225
226 /*****
227 *** the initial deal ***
228 *****/
229 // option fair price (Black-Scholes) at t=0
230 DiscountFactor rDiscount = std::exp(-r_*maturity_);
231 DiscountFactor qDiscount = std::exp(-stockDividendYield*maturity_);
232 Real forward = stock*qDiscount/rDiscount;
233 Real variance = sigma_*sigma_*maturity_;
234 boost::shared_ptr<StrikedTypePayoff> payoff(
235     new PlainVanillaPayoff(type_,strike_));
236 BlackFormula black(forward,rDiscount,variance,payoff);
237 // sell the option, cash in its premium
238 money_account += black.value();
239 // compute delta
240 Real delta = black.delta(stock);
241 // delta-hedge the option buying stock
242 Real stockAmount = delta;
243 money_account -= stockAmount*stock;
244
245 /*****
246 *** hedging during option life ***
247 *****/
248 for (Size step = 0; step < n-1; step++){
249
250     // time flows
251     t += dt;
252
253     // accruing on the money account
254     money_account *= std::exp( r_*dt );
255
256     // stock growth:
257     stock = path[step+1];
258
259     // recalculate option value at the current stock value,
260     // and the current time to maturity
261     rDiscount = std::exp(-r_*(maturity_-t));
262     qDiscount = std::exp(-stockDividendYield*(maturity_-t));
263     forward = stock*qDiscount/rDiscount;
264     variance = sigma_*sigma_*(maturity_-t);
265     BlackFormula black(forward,rDiscount,variance,payoff);
266
267     // recalculate delta
268     delta = black.delta(stock);
269
270     // re-hedging
271     money_account -= (delta - stockAmount)*stock;
272     stockAmount = delta;
273 }
274
275 /*****
276 *** option expiration ***
277 *****/
278 // last accrual on my money account
279 money_account *= std::exp( r_*dt );
280 // last stock growth

```

```

281     stock = path[n];
282
283     // the hedger delivers the option payoff to the option holder
284     Real optionPayoff = PlainVanillaPayoff(type_, strike_)(stock);
285     money_account -= optionPayoff;
286
287     // and unwinds the hedge selling his stock position
288     money_account += stockAmount*stock;
289
290     // final Profit&Loss
291     return money_account;
292 }
293
294
295 // The computation over nSamples paths of the P&L distribution
296 void ReplicationError::compute(Size nTimeSteps, Size nSamples)
297 {
298     QL_REQUIRE(nTimeSteps>0, "the number of steps must be > 0");
299
300     // hedging interval
301     // Time tau = maturity_ / nTimeSteps;
302
303     /* Black-Scholes framework: the underlying stock price evolves
304        lognormally with a fixed known volatility that stays constant
305        throughout time.
306     */
307     Date today = Date::todaysDate();
308     DayCounter dayCount = Actual365Fixed();
309     Handle<Quote> stateVariable(
310         boost::shared_ptr<Quote>(new SimpleQuote(s0_)));
311     Handle<YieldTermStructure> riskFreeRate(
312         boost::shared_ptr<YieldTermStructure>(
313             new FlatForward(today, r_, dayCount)));
314     Handle<YieldTermStructure> dividendYield(
315         boost::shared_ptr<YieldTermStructure>(
316             new FlatForward(today, 0.0, dayCount)));
317     Handle<BlackVolTermStructure> volatility(
318         boost::shared_ptr<BlackVolTermStructure>(
319             new BlackConstantVol(today, sigma_, dayCount)));
320     boost::shared_ptr<StochasticProcess1D> diffusion(
321         new BlackScholesProcess(stateVariable, dividendYield,
322             riskFreeRate, volatility));
323
324     // Black Scholes equation rules the path generator:
325     // at each step the log of the stock
326     // will have drift and sigma^2 variance
327     PseudoRandom::rsg_type rsg =
328         PseudoRandom::make_sequence_generator(nTimeSteps, 0);
329
330     bool brownianBridge = false;
331
332     typedef SingleVariate<PseudoRandom>::path_generator_type generator_type;
333     boost::shared_ptr<generator_type> myPathGenerator(new
334         generator_type(diffusion, maturity_, nTimeSteps,
335             rsg, brownianBridge));
336
337     // The replication strategy's Profit&Loss is computed for each path
338     // of the stock. The path pricer knows how to price a path using its
339     // value() method
340     boost::shared_ptr<PathPricer<Path> > myPathPricer(new
341         ReplicationPathPricer(payoff_.optionType(), payoff_.strike(),
342             r_, maturity_, sigma_));
343
344     // a statistics accumulator for the path-dependant Profit&Loss values
345     Statistics statisticsAccumulator;
346
347     // The OneFactorMontecarloModel generates paths using myPathGenerator

```

```

348 // each path is priced using myPathPricer
349 // prices will be accumulated into statisticsAccumulator
350 OneFactorMonteCarloOption MCSimulation(myPathGenerator,
351                                     myPathPricer,
352                                     statisticsAccumulator,
353                                     false);
354
355 // the model simulates nSamples paths
356 MCSimulation.addSamples(nSamples);
357
358 // the sampleAccumulator method of OneFactorMonteCarloOption_old
359 // gives access to all the methods of statisticsAccumulator
360 Real PLMean = MCSimulation.sampleAccumulator().mean();
361 Real PLStDev = MCSimulation.sampleAccumulator().standardDeviation();
362 Real PLSkew = MCSimulation.sampleAccumulator().skewness();
363 Real PLKurt = MCSimulation.sampleAccumulator().kurtosis();
364
365 // Derman and Kamal's formula
366 Real theorStD = std::sqrt(M_PI/4/nTimeSteps)*vega_*sigma_;
367
368
369 std::cout << std::fixed
370           << nSamples << "\t| "
371           << nTimeSteps << "\t | "
372           << std::setprecision(3) << PLMean << " \t| "
373           << std::setprecision(2) << PLStDev << " \t | "
374           << std::setprecision(2) << theorStD << " \t | "
375           << std::setprecision(2) << PLSkew << " \t| "
376           << std::setprecision(2) << PLKurt << std::endl;
377 }

```

## 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 #include <ql/quantlib.hpp>
21 #include <boost/timer.hpp>
22 #include <iostream>
23 #include <iomanip>
24
25 using namespace QuantLib;
26
27 #if defined(QL_ENABLE_SESSIONS)
28 namespace QuantLib {
29
30     Integer sessionId() { return 0; }
31
32 }
33 #endif
34
35
36 int main(int, char* [])
37 {
38     try {
39         QL_IO_INIT
40
41         boost::timer timer;
42         std::cout << std::endl;
43
44         // our options
45         Option::Type type(Option::Put);
46         Real underlying = 36;
47         Real strike = 40;
48         Spread dividendYield = 0.00;
49         Rate riskFreeRate = 0.06;
50         Volatility volatility = 0.20;
51
52         Date todaysDate(15, May, 1998);
53         Date settlementDate(17, May, 1998);
54         Settings::instance().evaluationDate() = todaysDate;
55
56         Date maturity(17, May, 1999);
57         DayCounter dayCounter = Actual365Fixed();
58
59         std::cout << "Option type = " << type << std::endl;
60         std::cout << "Maturity = " << maturity << std::endl;
61         std::cout << "Underlying price = " << underlying << std::endl;
62         std::cout << "Strike = " << strike << std::endl;
63         std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
64             << std::endl;
65         std::cout << "Dividend yield = " << io::rate(dividendYield)
66             << std::endl;
67         std::cout << "Volatility = " << io::volatility(volatility)
68             << std::endl;
69         std::cout << std::endl;
70
71         std::string method;
72
73         std::cout << std::endl ;
74
75         // write column headings
76         Size widths[] = { 35, 14, 14, 14 };
77         std::cout << std::setw(widths[0]) << std::left << "Method"
78             << std::setw(widths[1]) << std::left << "European"
79             << std::setw(widths[2]) << std::left << "Bermudan"

```



```

80         << std::setw(widths[3]) << std::left << "American"
81         << std::endl;
82
83     std::vector<Date> exerciseDates;
84     for (Integer i=1; i<=4; i++)
85         exerciseDates.push_back(settlementDate + 3*i*Months);
86
87     boost::shared_ptr<Exercise> europeanExercise(
88         new EuropeanExercise(maturity));
89
90     boost::shared_ptr<Exercise> bermudanExercise(
91         new BermudanExercise(exerciseDates));
92
93     boost::shared_ptr<Exercise> americanExercise(
94         new AmericanExercise(settlementDate,
95                               maturity));
96
97     Handle<Quote> underlyingH(
98         boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
99
100    // bootstrap the yield/dividend/vol curves
101    Handle<YieldTermStructure> flatTermStructure(
102        boost::shared_ptr<YieldTermStructure>(
103            new FlatForward(settlementDate, riskFreeRate, dayCounter)));
104    Handle<YieldTermStructure> flatDividendTS(
105        boost::shared_ptr<YieldTermStructure>(
106            new FlatForward(settlementDate, dividendYield, dayCounter)));
107    Handle<BlackVolTermStructure> flatVolTS(
108        boost::shared_ptr<BlackVolTermStructure>(
109            new BlackConstantVol(settlementDate, volatility, dayCounter)));
110
111    boost::shared_ptr<StrikedTypePayoff> payoff(new
112        PlainVanillaPayoff(type, strike));
113
114    boost::shared_ptr<BlackScholesProcess> stochasticProcess(new
115        BlackScholesProcess(
116            underlyingH,
117            flatDividendTS,
118            flatTermStructure,
119            flatVolTS));
120
121    // options
122
123    VanillaOption europeanOption(stochasticProcess, payoff,
124        europeanExercise);
125
126    VanillaOption bermudanOption(stochasticProcess, payoff,
127        bermudanExercise);
128
129    VanillaOption americanOption(stochasticProcess, payoff,
130        americanExercise);
131
132    // Analytic formulas:
133
134    // Black-Scholes for European
135    method = "Black-Scholes";
136    europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
137        new AnalyticEuropeanEngine));
138    std::cout << std::setw(widths[0]) << std::left << method
139        << std::fixed
140        << std::setw(widths[1]) << std::left << europeanOption.NPV()
141        << std::setw(widths[2]) << std::left << "N/A"
142        << std::setw(widths[3]) << std::left << "N/A"
143        << std::endl;
144
145    // Barone-Adesi and Whaley approximation for American
146    method = "Barone-Adesi/Whaley";

```

```

147     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
148         new BaroneAdesiWhaleyApproximationEngine));
149     std::cout << std::setw(widths[0]) << std::left << method
150         << std::fixed
151         << std::setw(widths[1]) << std::left << "N/A"
152         << std::setw(widths[2]) << std::left << "N/A"
153         << std::setw(widths[3]) << std::left << americanOption.NPV()
154         << std::endl;
155
156     // Bjerksund and Stensland approximation for American
157     method = "Bjerksund/Stensland";
158     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
159         new BjerksundStenslandApproximationEngine));
160     std::cout << std::setw(widths[0]) << std::left << method
161         << std::fixed
162         << std::setw(widths[1]) << std::left << "N/A"
163         << std::setw(widths[2]) << std::left << "N/A"
164         << std::setw(widths[3]) << std::left << americanOption.NPV()
165         << std::endl;
166
167     // Integral
168
169     method = "Integral";
170     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
171         new IntegralEngine));
172     std::cout << std::setw(widths[0]) << std::left << method
173         << std::fixed
174         << std::setw(widths[1]) << std::left << europeanOption.NPV()
175         << std::setw(widths[2]) << std::left << "N/A"
176         << std::setw(widths[3]) << std::left << "N/A"
177         << std::endl;
178
179     // Finite differences
180
181     Size timeSteps = 801;
182
183     method = "Finite differences";
184     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
185         new FDEuropeanEngine(timeSteps,timeSteps-1)));
186     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
187         new FDBermudanEngine(timeSteps,timeSteps-1)));
188     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
189         new FDAmericanEngine(timeSteps,timeSteps-1)));
190     std::cout << std::setw(widths[0]) << std::left << method
191         << std::fixed
192         << std::setw(widths[1]) << std::left << europeanOption.NPV()
193         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
194         << std::setw(widths[3]) << std::left << americanOption.NPV()
195         << std::endl;
196
197     // Binomial method
198
199     method = "Binomial Jarrow-Rudd";
200     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
201         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
202     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
203         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
204     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
205         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
206     std::cout << std::setw(widths[0]) << std::left << method
207         << std::fixed
208         << std::setw(widths[1]) << std::left << europeanOption.NPV()
209         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
210         << std::setw(widths[3]) << std::left << americanOption.NPV()
211         << std::endl;
212
213     method = "Binomial Cox-Ross-Rubinstein";

```

```

214     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
215         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
216     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
217         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
218     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
219         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
220     std::cout << std::setw(widths[0]) << std::left << method
221         << std::fixed
222         << std::setw(widths[1]) << std::left << europeanOption.NPV()
223         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
224         << std::setw(widths[3]) << std::left << americanOption.NPV()
225         << std::endl;
226
227     method = "Additive equiprobabilities";
228     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
229         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
230     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
231         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
232     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
233         new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
234     std::cout << std::setw(widths[0]) << std::left << method
235         << std::fixed
236         << std::setw(widths[1]) << std::left << europeanOption.NPV()
237         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
238         << std::setw(widths[3]) << std::left << americanOption.NPV()
239         << std::endl;
240
241     method = "Binomial Trigeorgis";
242     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
243         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
244     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
245         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
246     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
247         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
248     std::cout << std::setw(widths[0]) << std::left << method
249         << std::fixed
250         << std::setw(widths[1]) << std::left << europeanOption.NPV()
251         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
252         << std::setw(widths[3]) << std::left << americanOption.NPV()
253         << std::endl;
254
255     method = "Binomial Tian";
256     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
257         new BinomialVanillaEngine<Tian>(timeSteps)));
258     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
259         new BinomialVanillaEngine<Tian>(timeSteps)));
260     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
261         new BinomialVanillaEngine<Tian>(timeSteps)));
262     std::cout << std::setw(widths[0]) << std::left << method
263         << std::fixed
264         << std::setw(widths[1]) << std::left << europeanOption.NPV()
265         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
266         << std::setw(widths[3]) << std::left << americanOption.NPV()
267         << std::endl;
268
269     method = "Binomial Leisen-Reimer";
270     europeanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
271         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
272     bermudanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
273         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
274     americanOption.setPricingEngine(boost::shared_ptr<PricingEngine>(
275         new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
276     std::cout << std::setw(widths[0]) << std::left << method
277         << std::fixed
278         << std::setw(widths[1]) << std::left << europeanOption.NPV()
279         << std::setw(widths[2]) << std::left << bermudanOption.NPV()
280         << std::setw(widths[3]) << std::left << americanOption.NPV()

```

```

281         << std::endl;
282
283     // Monte Carlo Method
284
285     timeSteps = 1;
286
287     method = "MC (crude)";
288     Size mcSeed = 42;
289
290     boost::shared_ptr<PricingEngine> mcengine1;
291     mcengine1 =
292         MakeMCEuropeanEngine<PseudoRandom>().withSteps(timeSteps)
293                                             .withTolerance(0.02)
294                                             .withSeed(mcSeed);
295     europeanOption.setPricingEngine(mcengine1);
296     // Real errorEstimate = europeanOption.errorEstimate();
297     std::cout << std::setw(widths[0]) << std::left << method
298               << std::fixed
299               << std::setw(widths[1]) << std::left << europeanOption.NPV()
300               << std::setw(widths[2]) << std::left << "N/A"
301               << std::setw(widths[3]) << std::left << "N/A"
302               << std::endl;
303
304     method = "MC (Sobol)";
305     Size nSamples = 32768; // 2^15
306
307     boost::shared_ptr<PricingEngine> mcengine2;
308     mcengine2 =
309         MakeMCEuropeanEngine<LowDiscrepancy>().withSteps(timeSteps)
310                                             .withSamples(nSamples);
311     europeanOption.setPricingEngine(mcengine2);
312     std::cout << std::setw(widths[0]) << std::left << method
313               << std::fixed
314               << std::setw(widths[1]) << std::left << europeanOption.NPV()
315               << std::setw(widths[2]) << std::left << "N/A"
316               << std::setw(widths[3]) << std::left << "N/A"
317               << std::endl;
318
319     Real seconds = timer.elapsed();
320     Integer hours = int(seconds/3600);
321     seconds -= hours * 3600;
322     Integer minutes = int(seconds/60);
323     seconds -= minutes * 60;
324     std::cout << " \nRun completed in ";
325     if (hours > 0)
326         std::cout << hours << " h ";
327     if (hours > 0 || minutes > 0)
328         std::cout << minutes << " m ";
329     std::cout << std::fixed << std::setprecision(0)
330               << seconds << " s\n" << std::endl;
331
332     return 0;
333 } catch (std::exception& e) {
334     std::cout << e.what() << std::endl;
335     return 1;
336 } catch (...) {
337     std::cout << "unknown error" << std::endl;
338     return 1;
339 }
340 }
```

## 9.5 history\_iterators.cpp

This code exemplifies how to use History iterators to perform Gaussian statistic analyses on historical data.

```
1
2 // initialize a History
3 History h(...);
4
5 // print out the mean value and its standard deviation.
6
7 GaussianStatistics s;
8 s.addSequence(h.vdbegin(),h.vdend());
9 cout << "Historical mean: " << s.mean() << endl;
10 cout << "Std. deviation: " << s.standardDeviation() << endl;
11
12 // Another possibility: print out the maximum value.
13
14 History::const_valid_iterator max = h.vbegin(), i=max, end = h.vend();
15 for (i++; i!=end; i++)
16     if (i->value() > max->value())
17         max = i;
18 cout << "Maximum value: " << max->value()
19     << " assumed " << DateFormatter::toString(max->date()) << endl;
20
21 // or the minimum, this time the STL way:
22
23 bool lessthan(const History::Entry& i, const History::Entry& j) {
24     return i.value() < j.value();
25 }
26
27 History::const_valid_iterator min =
28     std::min_element(h.vbegin(),h.vend(),lessthan);
29 cout << "Minimum value: " << min->value()
30     << " assumed " << DateFormatter::toString(min->date()) << endl;
31
32
```

## 9.6 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
21 /* This example shows how to set up a Term Structure and then price a simple
22    swap.
23 */
24
25 // the only header you need to use QuantLib
26 #include <ql/quantlib.hpp>
27 #include <boost/timer.hpp>
28 #include <iostream>
29 #include <iomanip>
30
31 using namespace QuantLib;
32
33 #if defined(QL_ENABLE_SESSIONS)
34 namespace QuantLib {
35
36     Integer sessionId() { return 0; }
37
38 }
39 #endif
40
41
42 int main(int, char* [])
43 {
44     try {
45         QL_IO_INIT
46
47         boost::timer timer;
48         std::cout << std::endl;
49
50         /*****
51          *** MARKET DATA ***
52          *****/
53
54         Calendar calendar = TARGET();
55         // uncommenting the following line generates an error
56         // calendar = Tokyo();
57         Date settlementDate(22, September, 2004);
58         // must be a business day
59         settlementDate = calendar.adjust(settlementDate);
60
61         Integer fixingDays = 2;
62         Date todaysDate = calendar.advance(settlementDate, -fixingDays, Days);
63         // nothing to do with Date::todaysDate
64         Settings::instance().evaluationDate() = todaysDate;
65
66
67         todaysDate = Settings::instance().evaluationDate();
68         std::cout << "Today: " << todaysDate.weekday()
69                 << ", " << todaysDate << std::endl;
70
71         std::cout << "Settlement date: " << settlementDate.weekday()
72                 << ", " << settlementDate << std::endl;
73
74         // deposits
75         Rate d1wQuote=0.0382;
76         Rate d1mQuote=0.0372;
77         Rate d3mQuote=0.0363;
78         Rate d6mQuote=0.0353;
79         Rate d9mQuote=0.0348;
80         Rate d1yQuote=0.0345;

```

```

81      // FRAs
82      Rate fra3x6Quote=0.037125;
83      Rate fra6x9Quote=0.037125;
84      Rate fra6x12Quote=0.037125;
85      // futures
86      Real fut1Quote=96.2875;
87      Real fut2Quote=96.7875;
88      Real fut3Quote=96.9875;
89      Real fut4Quote=96.6875;
90      Real fut5Quote=96.4875;
91      Real fut6Quote=96.3875;
92      Real fut7Quote=96.2875;
93      Real fut8Quote=96.0875;
94      // swaps
95      Rate s2yQuote=0.037125;
96      Rate s3yQuote=0.0398;
97      Rate s5yQuote=0.0443;
98      Rate s10yQuote=0.05165;
99      Rate s15yQuote=0.055175;
100
101
102      /*****
103      ***   QUOTES   ***
104      *****/
105
106      // SimpleQuote stores a value which can be manually changed;
107      // other Quote subclasses could read the value from a database
108      // or some kind of data feed.
109
110      // deposits
111      boost::shared_ptr<Quote> d1wRate(new SimpleQuote(d1wQuote));
112      boost::shared_ptr<Quote> d1mRate(new SimpleQuote(d1mQuote));
113      boost::shared_ptr<Quote> d3mRate(new SimpleQuote(d3mQuote));
114      boost::shared_ptr<Quote> d6mRate(new SimpleQuote(d6mQuote));
115      boost::shared_ptr<Quote> d9mRate(new SimpleQuote(d9mQuote));
116      boost::shared_ptr<Quote> d1yRate(new SimpleQuote(d1yQuote));
117      // FRAs
118      boost::shared_ptr<Quote> fra3x6Rate(new SimpleQuote(fra3x6Quote));
119      boost::shared_ptr<Quote> fra6x9Rate(new SimpleQuote(fra6x9Quote));
120      boost::shared_ptr<Quote> fra6x12Rate(new SimpleQuote(fra6x12Quote));
121      // futures
122      boost::shared_ptr<Quote> fut1Price(new SimpleQuote(fut1Quote));
123      boost::shared_ptr<Quote> fut2Price(new SimpleQuote(fut2Quote));
124      boost::shared_ptr<Quote> fut3Price(new SimpleQuote(fut3Quote));
125      boost::shared_ptr<Quote> fut4Price(new SimpleQuote(fut4Quote));
126      boost::shared_ptr<Quote> fut5Price(new SimpleQuote(fut5Quote));
127      boost::shared_ptr<Quote> fut6Price(new SimpleQuote(fut6Quote));
128      boost::shared_ptr<Quote> fut7Price(new SimpleQuote(fut7Quote));
129      boost::shared_ptr<Quote> fut8Price(new SimpleQuote(fut8Quote));
130      // swaps
131      boost::shared_ptr<Quote> s2yRate(new SimpleQuote(s2yQuote));
132      boost::shared_ptr<Quote> s3yRate(new SimpleQuote(s3yQuote));
133      boost::shared_ptr<Quote> s5yRate(new SimpleQuote(s5yQuote));
134      boost::shared_ptr<Quote> s10yRate(new SimpleQuote(s10yQuote));
135      boost::shared_ptr<Quote> s15yRate(new SimpleQuote(s15yQuote));
136
137
138      /*****
139      ***   RATE HELPERS   ***
140      *****/
141
142      // RateHelpers are built from the above quotes together with
143      // other instrument dependant infos. Quotes are passed in
144      // relinkable handles which could be relinked to some other
145      // data source later.
146
147      // deposits

```

```

148     DayCounter depositDayCounter = Actual360();
149
150     boost::shared_ptr<RateHelper> d1w(new DepositRateHelper(
151         Handle<Quote>(d1wRate),
152         1, Weeks, fixingDays,
153         calendar, ModifiedFollowing, depositDayCounter));
154     boost::shared_ptr<RateHelper> d1m(new DepositRateHelper(
155         Handle<Quote>(d1mRate),
156         1, Months, fixingDays,
157         calendar, ModifiedFollowing, depositDayCounter));
158     boost::shared_ptr<RateHelper> d3m(new DepositRateHelper(
159         Handle<Quote>(d3mRate),
160         3, Months, fixingDays,
161         calendar, ModifiedFollowing, depositDayCounter));
162     boost::shared_ptr<RateHelper> d6m(new DepositRateHelper(
163         Handle<Quote>(d6mRate),
164         6, Months, fixingDays,
165         calendar, ModifiedFollowing, depositDayCounter));
166     boost::shared_ptr<RateHelper> d9m(new DepositRateHelper(
167         Handle<Quote>(d9mRate),
168         9, Months, fixingDays,
169         calendar, ModifiedFollowing, depositDayCounter));
170     boost::shared_ptr<RateHelper> d1y(new DepositRateHelper(
171         Handle<Quote>(d1yRate),
172         1, Years, fixingDays,
173         calendar, ModifiedFollowing, depositDayCounter));
174
175
176     // setup FRAs
177     boost::shared_ptr<RateHelper> fra3x6(new FraRateHelper(
178         Handle<Quote>(fra3x6Rate),
179         3, 6, fixingDays, calendar, ModifiedFollowing,
180         depositDayCounter));
181     boost::shared_ptr<RateHelper> fra6x9(new FraRateHelper(
182         Handle<Quote>(fra6x9Rate),
183         6, 9, fixingDays, calendar, ModifiedFollowing,
184         depositDayCounter));
185     boost::shared_ptr<RateHelper> fra6x12(new FraRateHelper(
186         Handle<Quote>(fra6x12Rate),
187         6, 12, fixingDays, calendar, ModifiedFollowing,
188         depositDayCounter));
189
190
191     // setup futures
192     Integer futMonths = 3;
193     Date imm = Date::nextIMMdate(settlementDate);
194     boost::shared_ptr<RateHelper> fut1(new FuturesRateHelper(
195         Handle<Quote>(fut1Price),
196         imm,
197         futMonths, calendar, ModifiedFollowing,
198         depositDayCounter));
199     imm = Date::nextIMMdate(imm+1);
200     boost::shared_ptr<RateHelper> fut2(new FuturesRateHelper(
201         Handle<Quote>(fut1Price),
202         imm,
203         futMonths, calendar, ModifiedFollowing,
204         depositDayCounter));
205     imm = Date::nextIMMdate(imm+1);
206     boost::shared_ptr<RateHelper> fut3(new FuturesRateHelper(
207         Handle<Quote>(fut1Price),
208         imm,
209         futMonths, calendar, ModifiedFollowing,
210         depositDayCounter));
211     imm = Date::nextIMMdate(imm+1);
212     boost::shared_ptr<RateHelper> fut4(new FuturesRateHelper(
213         Handle<Quote>(fut1Price),
214         imm,

```



```

215         futMonths, calendar, ModifiedFollowing,
216         depositDayCounter));
217     imm = Date::nextIMMdate(imm+1);
218     boost::shared_ptr<RateHelper> fut5(new FuturesRateHelper(
219         Handle<Quote>(fut1Price),
220         imm,
221         futMonths, calendar, ModifiedFollowing,
222         depositDayCounter));
223     imm = Date::nextIMMdate(imm+1);
224     boost::shared_ptr<RateHelper> fut6(new FuturesRateHelper(
225         Handle<Quote>(fut1Price),
226         imm,
227         futMonths, calendar, ModifiedFollowing,
228         depositDayCounter));
229     imm = Date::nextIMMdate(imm+1);
230     boost::shared_ptr<RateHelper> fut7(new FuturesRateHelper(
231         Handle<Quote>(fut1Price),
232         imm,
233         futMonths, calendar, ModifiedFollowing,
234         depositDayCounter));
235     imm = Date::nextIMMdate(imm+1);
236     boost::shared_ptr<RateHelper> fut8(new FuturesRateHelper(
237         Handle<Quote>(fut1Price),
238         imm,
239         futMonths, calendar, ModifiedFollowing,
240         depositDayCounter));
241
242
243     // setup swaps
244     Frequency swFixedLegFrequency = Annual;
245     BusinessDayConvention swFixedLegConvention = Unadjusted;
246     DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
247     Frequency swFloatingLegFrequency = Semiannual;
248     DayCounter swFloatingLegDayCounter = Actual360();
249
250     boost::shared_ptr<RateHelper> s2y(new SwapRateHelper(
251         Handle<Quote>(s2yRate),
252         2, Years, fixingDays,
253         calendar, swFixedLegFrequency,
254         swFixedLegConvention, swFixedLegDayCounter,
255         swFloatingLegFrequency, ModifiedFollowing,
256         swFloatingLegDayCounter));
257     boost::shared_ptr<RateHelper> s3y(new SwapRateHelper(
258         Handle<Quote>(s3yRate),
259         3, Years, fixingDays,
260         calendar, swFixedLegFrequency,
261         swFixedLegConvention, swFixedLegDayCounter,
262         swFloatingLegFrequency, ModifiedFollowing,
263         swFloatingLegDayCounter));
264     boost::shared_ptr<RateHelper> s5y(new SwapRateHelper(
265         Handle<Quote>(s5yRate),
266         5, Years, fixingDays,
267         calendar, swFixedLegFrequency,
268         swFixedLegConvention, swFixedLegDayCounter,
269         swFloatingLegFrequency, ModifiedFollowing,
270         swFloatingLegDayCounter));
271     boost::shared_ptr<RateHelper> s10y(new SwapRateHelper(
272         Handle<Quote>(s10yRate),
273         10, Years, fixingDays,
274         calendar, swFixedLegFrequency,
275         swFixedLegConvention, swFixedLegDayCounter,
276         swFloatingLegFrequency, ModifiedFollowing,
277         swFloatingLegDayCounter));
278     boost::shared_ptr<RateHelper> s15y(new SwapRateHelper(
279         Handle<Quote>(s15yRate),
280         15, Years, fixingDays,
281         calendar, swFixedLegFrequency,

```

```

282         swFixedLegConvention, swFixedLegDayCounter,
283         swFloatingLegFrequency, ModifiedFollowing,
284         swFloatingLegDayCounter));
285
286
287     /*****
288     **  CURVE BUILDING **
289     *****/
290
291     // Any DayCounter would be fine.
292     // ActualActual::ISDA ensures that 30 years is 30.0
293     DayCounter termStructureDayCounter =
294         ActualActual(ActualActual::ISDA);
295
296
297     double tolerance = 1.0e-15;
298
299     // A depo-swap curve
300     std::vector<boost::shared_ptr<RateHelper> > depoSwapInstruments;
301     depoSwapInstruments.push_back(d1w);
302     depoSwapInstruments.push_back(d1m);
303     depoSwapInstruments.push_back(d3m);
304     depoSwapInstruments.push_back(d6m);
305     depoSwapInstruments.push_back(d9m);
306     depoSwapInstruments.push_back(d1y);
307     depoSwapInstruments.push_back(s2y);
308     depoSwapInstruments.push_back(s3y);
309     depoSwapInstruments.push_back(s5y);
310     depoSwapInstruments.push_back(s10y);
311     depoSwapInstruments.push_back(s15y);
312     boost::shared_ptr<YieldTermStructure> depoSwapTermStructure(new
313         PiecewiseFlatForward(settlementDate, depoSwapInstruments,
314                             termStructureDayCounter, tolerance));
315
316
317     // A depo-futures-swap curve
318     std::vector<boost::shared_ptr<RateHelper> > depoFutSwapInstruments;
319     depoFutSwapInstruments.push_back(d1w);
320     depoFutSwapInstruments.push_back(d1m);
321     depoFutSwapInstruments.push_back(fut1);
322     depoFutSwapInstruments.push_back(fut2);
323     depoFutSwapInstruments.push_back(fut3);
324     depoFutSwapInstruments.push_back(fut4);
325     depoFutSwapInstruments.push_back(fut5);
326     depoFutSwapInstruments.push_back(fut6);
327     depoFutSwapInstruments.push_back(fut7);
328     depoFutSwapInstruments.push_back(fut8);
329     depoFutSwapInstruments.push_back(s3y);
330     depoFutSwapInstruments.push_back(s5y);
331     depoFutSwapInstruments.push_back(s10y);
332     depoFutSwapInstruments.push_back(s15y);
333     boost::shared_ptr<YieldTermStructure> depoFutSwapTermStructure(new
334         PiecewiseFlatForward(settlementDate, depoFutSwapInstruments,
335                             termStructureDayCounter, tolerance));
336
337
338     // A depo-FRA-swap curve
339     std::vector<boost::shared_ptr<RateHelper> > depoFRASwapInstruments;
340     depoFRASwapInstruments.push_back(d1w);
341     depoFRASwapInstruments.push_back(d1m);
342     depoFRASwapInstruments.push_back(d3m);
343     depoFRASwapInstruments.push_back(fra3x6);
344     depoFRASwapInstruments.push_back(fra6x9);
345     depoFRASwapInstruments.push_back(fra6x12);
346     depoFRASwapInstruments.push_back(s2y);
347     depoFRASwapInstruments.push_back(s3y);
348     depoFRASwapInstruments.push_back(s5y);

```

```

349     depoFRASwapInstruments.push_back(s10y);
350     depoFRASwapInstruments.push_back(s15y);
351     boost::shared_ptr<YieldTermStructure> depoFRASwapTermStructure(new
352         PiecewiseFlatForward(settlementDate, depoFRASwapInstruments,
353             termStructureDayCounter, tolerance));
354
355
356     // Term structures that will be used for pricing:
357     // the one used for discounting cash flows
358     Handle<YieldTermStructure> discountingTermStructure;
359     // the one used for forward rate forecasting
360     Handle<YieldTermStructure> forecastingTermStructure;
361
362
363     /*****
364     * SWAPS TO BE PRICED *
365     *****/
366
367     // constant nominal 1,000,000 Euro
368     Real nominal = 1000000.0;
369     // fixed leg
370     Frequency fixedLegFrequency = Annual;
371     BusinessDayConvention fixedLegConvention = Unadjusted;
372     BusinessDayConvention floatingLegConvention = ModifiedFollowing;
373     DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
374     Rate fixedRate = 0.04;
375     DayCounter floatingLegDayCounter = Actual360();
376
377     // floating leg
378     Frequency floatingLegFrequency = Semiannual;
379     boost::shared_ptr<Xibor> euriborIndex(new Euribor(6, Months,
380         forecastingTermStructure)); // using the forecasting curve
381     Spread spread = 0.0;
382
383     Integer lenghtInYears = 5;
384     bool payFixedRate = true;
385
386     Date maturity = calendar.advance(settlementDate, lenghtInYears, Years,
387         floatingLegConvention);
388     Schedule fixedSchedule(calendar, settlementDate, maturity,
389         fixedLegFrequency, fixedLegConvention);
390     Schedule floatSchedule(calendar, settlementDate, maturity,
391         floatingLegFrequency, floatingLegConvention);
392     VanillaSwap spot5YearSwap(
393         payFixedRate, nominal,
394         fixedSchedule, fixedRate, fixedLegDayCounter,
395         floatSchedule, euriborIndex, fixingDays, spread,
396         floatingLegDayCounter, discountingTermStructure);
397
398     Date fwdStart = calendar.advance(settlementDate, 1, Years);
399     Date fwdMaturity = calendar.advance(fwdStart, lenghtInYears, Years,
400         floatingLegConvention);
401     Schedule fwdFixedSchedule(calendar, fwdStart, fwdMaturity,
402         fixedLegFrequency, fixedLegConvention);
403     Schedule fwdFloatSchedule(calendar, fwdStart, fwdMaturity,
404         floatingLegFrequency, floatingLegConvention);
405     VanillaSwap oneYearForward5YearSwap(
406         payFixedRate, nominal,
407         fwdFixedSchedule, fixedRate, fixedLegDayCounter,
408         fwdFloatSchedule, euriborIndex, fixingDays, spread,
409         floatingLegDayCounter, discountingTermStructure);
410
411
412     /*****
413     * SWAP PRICING *
414     *****/
415

```

```

416 // utilities for reporting
417 std::vector<std::string> headers(4);
418 headers[0] = "term structure";
419 headers[1] = "net present value";
420 headers[2] = "fair spread";
421 headers[3] = "fair fixed rate";
422 std::string separator = " | ";
423 Size width = headers[0].size() + separator.size()
424             + headers[1].size() + separator.size()
425             + headers[2].size() + separator.size()
426             + headers[3].size() + separator.size() - 1;
427 std::string rule(width, '-'), dblrule(width, '=');
428 std::string tab(8, ' ');
429
430 // calculations
431
432 std::cout << dblrule << std::endl;
433 std::cout << "5-year market swap-rate = "
434             << std::setprecision(2) << io::rate(s5yRate->value())
435             << std::endl;
436 std::cout << dblrule << std::endl;
437
438 std::cout << tab << "5-years swap paying "
439             << io::rate(fixedRate) << std::endl;
440 std::cout << headers[0] << separator
441             << headers[1] << separator
442             << headers[2] << separator
443             << headers[3] << separator << std::endl;
444 std::cout << rule << std::endl;
445
446 Real NPV;
447 Rate fairRate;
448 Spread fairSpread;
449
450 // Of course, you're not forced to really use different curves
451 forecastingTermStructure.linkTo(depoSwapTermStructure);
452 discountingTermStructure.linkTo(depoSwapTermStructure);
453
454 NPV = spot5YearSwap.NPV();
455 fairSpread = spot5YearSwap.fairSpread();
456 fairRate = spot5YearSwap.fairRate();
457
458 std::cout << std::setw(headers[0].size())
459             << "depo-swap" << separator;
460 std::cout << std::setw(headers[1].size())
461             << std::fixed << std::setprecision(2) << NPV << separator;
462 std::cout << std::setw(headers[2].size())
463             << io::rate(fairSpread) << separator;
464 std::cout << std::setw(headers[3].size())
465             << io::rate(fairRate) << separator;
466 std::cout << std::endl;
467
468
469 // let's check that the 5 years swap has been correctly re-priced
470 QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
471            "5-years swap mispriced by "
472            << io::rate(std::fabs(fairRate-s5yQuote)));
473
474
475 forecastingTermStructure.linkTo(depoFutSwapTermStructure);
476 discountingTermStructure.linkTo(depoFutSwapTermStructure);
477
478 NPV = spot5YearSwap.NPV();
479 fairSpread = spot5YearSwap.fairSpread();
480 fairRate = spot5YearSwap.fairRate();
481
482 std::cout << std::setw(headers[0].size())

```

```

483         << "depo-fut-swap" << separator;
484     std::cout << std::setw(headers[1].size())
485         << std::fixed << std::setprecision(2) << NPV << separator;
486     std::cout << std::setw(headers[2].size())
487         << io::rate(fairSpread) << separator;
488     std::cout << std::setw(headers[3].size())
489         << io::rate(fairRate) << separator;
490     std::cout << std::endl;
491
492     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
493         "5-years swap mispriced!");
494
495
496     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
497     discountingTermStructure.linkTo(depoFRASwapTermStructure);
498
499     NPV = spot5YearSwap.NPV();
500     fairSpread = spot5YearSwap.fairSpread();
501     fairRate = spot5YearSwap.fairRate();
502
503     std::cout << std::setw(headers[0].size())
504         << "depo-FRA-swap" << separator;
505     std::cout << std::setw(headers[1].size())
506         << std::fixed << std::setprecision(2) << NPV << separator;
507     std::cout << std::setw(headers[2].size())
508         << io::rate(fairSpread) << separator;
509     std::cout << std::setw(headers[3].size())
510         << io::rate(fairRate) << separator;
511     std::cout << std::endl;
512
513     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
514         "5-years swap mispriced!");
515
516
517     std::cout << rule << std::endl;
518
519     // now let's price the 1Y forward 5Y swap
520
521     std::cout << tab << "5-years, 1-year forward swap paying "
522         << io::rate(fixedRate) << std::endl;
523     std::cout << headers[0] << separator
524         << headers[1] << separator
525         << headers[2] << separator
526         << headers[3] << separator << std::endl;
527     std::cout << rule << std::endl;
528
529
530     forecastingTermStructure.linkTo(depoSwapTermStructure);
531     discountingTermStructure.linkTo(depoSwapTermStructure);
532
533     NPV = oneYearForward5YearSwap.NPV();
534     fairSpread = oneYearForward5YearSwap.fairSpread();
535     fairRate = oneYearForward5YearSwap.fairRate();
536
537     std::cout << std::setw(headers[0].size())
538         << "depo-swap" << separator;
539     std::cout << std::setw(headers[1].size())
540         << std::fixed << std::setprecision(2) << NPV << separator;
541     std::cout << std::setw(headers[2].size())
542         << io::rate(fairSpread) << separator;
543     std::cout << std::setw(headers[3].size())
544         << io::rate(fairRate) << separator;
545     std::cout << std::endl;
546
547
548     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
549     discountingTermStructure.linkTo(depoFutSwapTermStructure);

```

```

550
551     NPV = oneYearForward5YearSwap.NPV();
552     fairSpread = oneYearForward5YearSwap.fairSpread();
553     fairRate = oneYearForward5YearSwap.fairRate();
554
555     std::cout << std::setw(headers[0].size())
556               << "depo-fut-swap" << separator;
557     std::cout << std::setw(headers[1].size())
558               << std::fixed << std::setprecision(2) << NPV << separator;
559     std::cout << std::setw(headers[2].size())
560               << io::rate(fairSpread) << separator;
561     std::cout << std::setw(headers[3].size())
562               << io::rate(fairRate) << separator;
563     std::cout << std::endl;
564
565
566     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
567     discountingTermStructure.linkTo(depoFRASwapTermStructure);
568
569     NPV = oneYearForward5YearSwap.NPV();
570     fairSpread = oneYearForward5YearSwap.fairSpread();
571     fairRate = oneYearForward5YearSwap.fairRate();
572
573     std::cout << std::setw(headers[0].size())
574               << "depo-FRA-swap" << separator;
575     std::cout << std::setw(headers[1].size())
576               << std::fixed << std::setprecision(2) << NPV << separator;
577     std::cout << std::setw(headers[2].size())
578               << io::rate(fairSpread) << separator;
579     std::cout << std::setw(headers[3].size())
580               << io::rate(fairRate) << separator;
581     std::cout << std::endl;
582
583
584     // now let's say that the 5-years swap rate goes up to 4.60%.
585     // A smarter market element--say, connected to a data source-- would
586     // notice the change itself. Since we're using SimpleQuotes,
587     // we'll have to change the value manually--which forces us to
588     // downcast the handle and use the SimpleQuote
589     // interface. In any case, the point here is that a change in the
590     // value contained in the Quote triggers a new bootstrapping
591     // of the curve and a repricing of the swap.
592
593     boost::shared_ptr<SimpleQuote> fiveYearsRate =
594         boost::dynamic_pointer_cast<SimpleQuote>(s5yRate);
595     fiveYearsRate->setValue(0.0460);
596
597     std::cout << dblrule << std::endl;
598     std::cout << "5-year market swap-rate = "
599               << io::rate(s5yRate->value()) << std::endl;
600     std::cout << dblrule << std::endl;
601
602     std::cout << tab << "5-years swap paying "
603               << io::rate(fixedRate) << std::endl;
604     std::cout << headers[0] << separator
605               << headers[1] << separator
606               << headers[2] << separator
607               << headers[3] << separator << std::endl;
608     std::cout << rule << std::endl;
609
610     // now get the updated results
611     forecastingTermStructure.linkTo(depoSwapTermStructure);
612     discountingTermStructure.linkTo(depoSwapTermStructure);
613
614     NPV = spot5YearSwap.NPV();
615     fairSpread = spot5YearSwap.fairSpread();
616     fairRate = spot5YearSwap.fairRate();

```

```

617
618     std::cout << std::setw(headers[0].size())
619               << "depo-swap" << separator;
620     std::cout << std::setw(headers[1].size())
621               << std::fixed << std::setprecision(2) << NPV << separator;
622     std::cout << std::setw(headers[2].size())
623               << io::rate(fairSpread) << separator;
624     std::cout << std::setw(headers[3].size())
625               << io::rate(fairRate) << separator;
626     std::cout << std::endl;
627
628     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
629               "5-years swap mispriced!");
630
631
632     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
633     discountingTermStructure.linkTo(depoFutSwapTermStructure);
634
635     NPV = spot5YearSwap.NPV();
636     fairSpread = spot5YearSwap.fairSpread();
637     fairRate = spot5YearSwap.fairRate();
638
639     std::cout << std::setw(headers[0].size())
640               << "depo-fut-swap" << separator;
641     std::cout << std::setw(headers[1].size())
642               << std::fixed << std::setprecision(2) << NPV << separator;
643     std::cout << std::setw(headers[2].size())
644               << io::rate(fairSpread) << separator;
645     std::cout << std::setw(headers[3].size())
646               << io::rate(fairRate) << separator;
647     std::cout << std::endl;
648
649     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
650               "5-years swap mispriced!");
651
652
653     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
654     discountingTermStructure.linkTo(depoFRASwapTermStructure);
655
656     NPV = spot5YearSwap.NPV();
657     fairSpread = spot5YearSwap.fairSpread();
658     fairRate = spot5YearSwap.fairRate();
659
660     std::cout << std::setw(headers[0].size())
661               << "depo-FRA-swap" << separator;
662     std::cout << std::setw(headers[1].size())
663               << std::fixed << std::setprecision(2) << NPV << separator;
664     std::cout << std::setw(headers[2].size())
665               << io::rate(fairSpread) << separator;
666     std::cout << std::setw(headers[3].size())
667               << io::rate(fairRate) << separator;
668     std::cout << std::endl;
669
670     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
671               "5-years swap mispriced!");
672
673     std::cout << rule << std::endl;
674
675     // the 1Y forward 5Y swap changes as well
676
677     std::cout << tab << "5-years, 1-year forward swap paying "
678               << io::rate(fixedRate) << std::endl;
679     std::cout << headers[0] << separator
680               << headers[1] << separator
681               << headers[2] << separator
682               << headers[3] << separator << std::endl;
683     std::cout << rule << std::endl;

```

```

684
685
686     forecastingTermStructure.linkTo(depoSwapTermStructure);
687     discountingTermStructure.linkTo(depoSwapTermStructure);
688
689     NPV = oneYearForward5YearSwap.NPV();
690     fairSpread = oneYearForward5YearSwap.fairSpread();
691     fairRate = oneYearForward5YearSwap.fairRate();
692
693     std::cout << std::setw(headers[0].size())
694               << "depo-swap" << separator;
695     std::cout << std::setw(headers[1].size())
696               << std::fixed << std::setprecision(2) << NPV << separator;
697     std::cout << std::setw(headers[2].size())
698               << io::rate(fairSpread) << separator;
699     std::cout << std::setw(headers[3].size())
700               << io::rate(fairRate) << separator;
701     std::cout << std::endl;
702
703
704     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
705     discountingTermStructure.linkTo(depoFutSwapTermStructure);
706
707     NPV = oneYearForward5YearSwap.NPV();
708     fairSpread = oneYearForward5YearSwap.fairSpread();
709     fairRate = oneYearForward5YearSwap.fairRate();
710
711     std::cout << std::setw(headers[0].size())
712               << "depo-fut-swap" << separator;
713     std::cout << std::setw(headers[1].size())
714               << std::fixed << std::setprecision(2) << NPV << separator;
715     std::cout << std::setw(headers[2].size())
716               << io::rate(fairSpread) << separator;
717     std::cout << std::setw(headers[3].size())
718               << io::rate(fairRate) << separator;
719     std::cout << std::endl;
720
721
722     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
723     discountingTermStructure.linkTo(depoFRASwapTermStructure);
724
725     NPV = oneYearForward5YearSwap.NPV();
726     fairSpread = oneYearForward5YearSwap.fairSpread();
727     fairRate = oneYearForward5YearSwap.fairRate();
728
729     std::cout << std::setw(headers[0].size())
730               << "depo-FRA-swap" << separator;
731     std::cout << std::setw(headers[1].size())
732               << std::fixed << std::setprecision(2) << NPV << separator;
733     std::cout << std::setw(headers[2].size())
734               << io::rate(fairSpread) << separator;
735     std::cout << std::setw(headers[3].size())
736               << io::rate(fairRate) << separator;
737     std::cout << std::endl;
738
739     Real seconds = timer.elapsed();
740     Integer hours = int(seconds/3600);
741     seconds -= hours * 3600;
742     Integer minutes = int(seconds/60);
743     seconds -= minutes * 60;
744     std::cout << " \nRun completed in ";
745     if (hours > 0)
746         std::cout << hours << " h ";
747     if (hours > 0 || minutes > 0)
748         std::cout << minutes << " m ";
749     std::cout << std::fixed << std::setprecision(0)
750               << seconds << " s\n" << std::endl;

```



```
751
752     return 0;
753
754     } catch (std::exception& e) {
755         std::cout << e.what() << std::endl;
756         return 1;
757     } catch (...) {
758         std::cout << "unknown error" << std::endl;
759         return 1;
760     }
761 }
762
```

## 9.7 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

unlike in previous versions, the returned vector now include 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 `ConvertibleFixedCouponBond`**

At this time, discrete dividends are not managed.

**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`**

At this time, discrete dividends are not managed.

**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`**

At this time, discrete dividends are not managed.

**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 &`  
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`)

The result date is following or equal to the original 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 `DepositRateHelper`**

This class assumes that the reference date does not change between calls of `setTermStructure()`.

**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()`, `Ex`  
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 &c)`**  
 being this a rollback, from must be a later time than to.

**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.

**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 `FraRateHelper`**

This class assumes that the reference date does not change between calls of `setTermStructure()`.

**Class `FuturesRateHelper`**

This class assumes that the reference date does not change between calls of `setTermStructure()`.

**Class `G2SwaptionEngine`**

The engine assumes that the exercise date equals the start date of the passed swap.

**Member `Handle::Handle(const boost::shared_ptr< Type > &h=boostshared_ptr< Type >(), bool registerAsObserver)`**  
 see the documentation of the `Link` class for issues relatives to `registerAsObserver`.

**Member `Handle::linkTo(const boost::shared_ptr< Type > &, 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/standardDeviation 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.

**Class `IndexedCoupon`**

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.

**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 InterestRate'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< Type > &h=boostshared\_ptr< Type >(), bool registerAsObserver=**

see the documentation of the [linkTo\(\)](#) method for issues relatives to [registerAsObserver](#).

**Member [Link::linkTo](#)(const boost::shared\_ptr< Type > &, 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 [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:

**Member [OneAssetOption::impliedVolatility](#)(Real price, Real accuracy=1.0e-4, Size maxEvaluations=100, Volatil**

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, Volatil**

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.

**Class `SwapRateHelper`**

This class assumes that the settlement date does not change between calls of `setTermStructure()`.

**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.

**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 [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 `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.

**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 [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 [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 [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 [SequenceStatistics](#)**

the correctness of the returned values is tested by checking them against numerical calculations.

**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.

**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 earliestDate 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.

**Member [BlackScholesProcess::drift](#)(Time t, Real x) const**

revise extrapolation

**Member [BlackScholesProcess::diffusion](#)(Time t, Real x) const**

revise extrapolation

**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

**Class [EarlyExercise](#)**

derive a plain American Exercise class (no earliestDate, no payoffAtExpiry)

**Member [Event::hasOccurred](#)(const Date &d, bool includeToday=false) const**

make QL\_TODAYS\_PAYMENT dynamically configurable?

**Class [ExplicitEuler](#)**

add Richardson extrapolation

**Class [FraRateHelper](#)**

convexity adjustment should be implemented.

**Class [GenericRiskStatistics](#)**

add historical annualized volatility

**Class [IntegralEngine](#)**

define tolerance for calculate()

**Class [Jibar](#)**

check settlement days and day-count convention.

**Class [LogLinearInterpolation](#)**

implement primitive, derivative, and secondDerivative functions.

**Member [pseudoSqrt](#)**

- implement Hypersphere decomposition:
  1. Jäckel "Monte Carlo Methods in Finance", Chapter 6
  2. Brigo "A Note on Correlation and Rank Reduction"
  3. Rapisarda, Brigo, Mercurio "Parameterizing correlations: a geometric interpretation"
- implement Higham algorithm: Higham "Computing the nearest correlation matrix"

**Class [McDiscreteArithmeticASO](#)**

continous-averaging version

**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 [SwapRateHelper](#)**

currency and day counter of Xibor should be added to obtain well-defined `SwapRateHelper`

**Class [Swaption](#)**

add explicit exercise lag

**Class [SwaptionVolatilityMatrix](#)**

either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the exercise date and length vector but an interpolation pointing to the original ones.

**Class [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 [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 [BPSBasketCalculator](#)

this class must still be checked. It is not guaranteed to yield the right results.

### 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 [MCAmericanBasketEngine](#)**

This method is intrinsically weak for out-of-the-money options.

**Class [MCAmericanBasketEngine](#)**

this engine does not yet work for put options. More problems might surface.

**Class [MultiCubicSpline](#)**

- cannot interpolate at the grid points on the boundary surface of the N-dimensional region
- it does not compile under Borland

**Member [Swap::sensitivity\(Integer basis=2\) const](#)**

this method is not guaranteed to yield the right results.

**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::businessDayConvention\(\)](#) const

use either paymentConvention() or accrualConvention()

Member [Bond::Bond](#)(const DayCounter &dayCount, const Calendar &calendar, BusinessDayConvention businessDayConvention) const

use the other constructor

Member [FixedCouponBond::FixedCouponBond](#)(const Date &issueDate, const Date &datedDate, const Date &maturityDate) const

use the other constructor

Member [FixedCouponBondHelper::FixedCouponBondHelper](#)(const Handle< Quote > &cleanPrice, const Date &issueDate, const Date &datedDate, const Date &maturityDate) const

use the other constructor

Member [FloatingRateBond::FloatingRateBond](#)(const Date &issueDate, const Date &datedDate, const Date &maturityDate, const Rate &rate) const

use the other constructor

Member [Swap::sensitivity](#)(Integer basis=2) const

this method will be removed in future releases.

Member [SwapRateHelper::SwapRateHelper](#)(const Handle< Quote > &rate, Integer n, TimeUnit units, Integer settlementDays, const Date &startDate, const Date &endDate) const

use one of the other constructors

Member [SwapRateHelper::SwapRateHelper](#)(Rate rate, Integer n, TimeUnit units, Integer settlementDays, const Date &startDate, const Date &endDate) const

use one of the other constructors

Member [SwaptionHelper::SwaptionHelper](#)(const Period &maturity, const Period &length, const Handle< Quote > &rate, const Date &startDate, const Date &endDate) const

use the other constructor

Member [TimeGrid::findIndex](#)(Time t) const

use index() instead

Member [UnitedStates::Exchange](#)

use NYSE instead

Member [VanillaSwap::VanillaSwap](#)(bool payFixedRate, Real nominal, const Schedule &fixedSchedule, Rate fixedRate, const Date &startDate, const Date &endDate) const

use the other constructor

# Index

- ~Error
  - QuantLib::Error, [400](#)
- accruedAmount
  - QuantLib::Bond, [245](#)
- add
  - QuantLib::ExchangeRateManager, [414](#)
  - QuantLib::GeneralStatistics, [499](#)
  - QuantLib::IncrementalStatistics, [545](#)
- addHoliday
  - QuantLib::Calendar, [263](#)
- addSequence
  - QuantLib::IncrementalStatistics, [545](#)
- adjust
  - QuantLib::Calendar, [264](#)
- adjustValues
  - QuantLib::DiscretizedAsset, [372](#)
- advance
  - QuantLib::Calendar, [264](#)
- amount
  - QuantLib::CashFlow, [285](#)
  - QuantLib::Dividend, [379](#)
  - QuantLib::FixedDividend, [442](#)
  - QuantLib::FixedRateCoupon, [445](#)
  - QuantLib::FractionalDividend, [466](#)
  - QuantLib::IndexedCoupon, [548](#)
  - QuantLib::ParCoupon, [746](#)
  - QuantLib::Short, [805](#)
  - QuantLib::SimpleCashFlow, [810](#)
- Annual
  - datetime, [95](#)
- apply
  - QuantLib::BlackScholesProcess, [229](#)
  - QuantLib::HestonProcess, [516](#)
  - QuantLib::LiborForwardModelProcess, [620](#)
  - QuantLib::Merton76Process, [677](#)
  - QuantLib::StochasticProcess, [841](#)
  - QuantLib::StochasticProcess1D, [843](#)
  - QuantLib::StochasticProcessArray, [847](#)
- applyAfterApplying
  - QuantLib::BoundaryCondition, [246](#)
  - QuantLib::DirichletBC, [362](#)
  - QuantLib::NeumannBC, [700](#)
- applyAfterSolving
  - QuantLib::BoundaryCondition, [247](#)
  - QuantLib::DirichletBC, [362](#)
  - QuantLib::NeumannBC, [701](#)
- applyBeforeApplying
  - QuantLib::BoundaryCondition, [246](#)
  - QuantLib::DirichletBC, [362](#)
  - QuantLib::NeumannBC, [700](#)
- applyBeforeSolving
  - QuantLib::BoundaryCondition, [246](#)
  - QuantLib::DirichletBC, [362](#)
  - QuantLib::NeumannBC, [700](#)
- Asian option engines, [101](#)
- AutomatedConversion
  - QuantLib::Money, [683](#)
- averageShortfall
  - QuantLib::GenericRiskStatistics, [504](#)
- BackwardFlatInterpolation
  - QuantLib::BackwardFlatInterpolation, [188](#)
- Barrier option engines, [102](#)
- BaseCurrencyConversion
  - QuantLib::Money, [683](#)
- Basket option engines, [103](#)
- BEJ
  - QuantLib::Indonesia, [553](#)
- BicubicSpline
  - QuantLib::BicubicSpline, [207](#)
- BilinearInterpolation
  - QuantLib::BilinearInterpolation, [209](#)
- Bimonthly
  - datetime, [95](#)
- blackVarianceImpl
  - QuantLib::BlackVolatilityTermStructure, [238](#)
- BlackVarianceTermStructure
  - QuantLib::BlackVarianceTermStructure, [236](#)
- BlackVolatilityTermStructure
  - QuantLib::BlackVolatilityTermStructure, [238](#)
- blackVolImpl
  - QuantLib::BlackVarianceTermStructure, [236](#)
- BlackVolTermStructure
  - QuantLib::BlackVolTermStructure, [241](#)

- BMV
  - QuantLib::Mexico, [679](#)
- Bond
  - QuantLib::Bond, [244](#)
- BoundaryCondition
  - QuantLib::CubicSpline, [337](#)
- bps
  - QuantLib::Cashflows, [288](#)
- BSMTermOperator
  - findiff, [118](#)
- BSSE
  - QuantLib::Slovakia, [825](#)
- BusinessDayConvention
  - datetime, [94](#)
- businessDayConvention
  - QuantLib::Bond, [244](#)
- calculate
  - QuantLib::Instrument, [556](#)
  - QuantLib::LazyObject, [604](#)
  - QuantLib::McSimulation, [673](#)
- Calendar
  - QuantLib::Calendar, [263](#)
- Calendars, [96](#)
- calibrate
  - QuantLib::ShortRateModel, [808](#)
- Cap/floor engines, [104](#)
- CapletVolatilityStructure
  - QuantLib::CapletVolatilityStructure, [280](#)
- CapVolatilityStructure
  - QuantLib::CapVolatilityStructure, [282](#)
- cashflows
  - QuantLib::Bond, [244](#)
- Ceiling
  - QuantLib::Rounding, [788](#)
- cleanPrice
  - QuantLib::Bond, [244](#)
- Cliquet option engines, [105](#)
- Closest
  - QuantLib::Rounding, [788](#)
- compoundFactor
  - QuantLib::InterestRate, [560](#)
- compoundForwardImpl
  - QuantLib::ExtendedDiscountCurve, [424](#)
- ConversionType
  - QuantLib::Money, [683](#)
- convexity
  - QuantLib::Cashflows, [289](#)
- correlationMatrix
  - QuantLib::CovarianceDecomposition, [328](#)
- Coupon
  - QuantLib::Coupon, [327](#)
- covariance
  - QuantLib::EulerDiscretization, [404](#)
  - QuantLib::LiborForwardModelProcess, [620](#)
  - QuantLib::StochasticProcess, [840](#)
  - QuantLib::StochasticProcessArray, [847](#)
- CovarianceDecomposition
  - QuantLib::CovarianceDecomposition, [328](#)
- CubicSpline
  - QuantLib::CubicSpline, [337](#)
- Currencies and FX rates, [89](#)
- Currency
  - QuantLib::Currency, [345](#)
- date
  - QuantLib::CashFlow, [285](#)
  - QuantLib::Coupon, [327](#)
  - QuantLib::Dividend, [378](#)
  - QuantLib::SimpleCashFlow, [810](#)
- Date and time calculations, [93](#)
- datetime
  - Annual, [95](#)
  - Bimonthly, [95](#)
  - BusinessDayConvention, [94](#)
  - EveryFourthMonth, [95](#)
  - Following, [94](#)
  - Frequency, [95](#)
  - ModifiedFollowing, [95](#)
  - ModifiedPreceding, [94](#)
  - MonthEndReference, [95](#)
  - Monthly, [95](#)
  - NoFrequency, [95](#)
  - Once, [95](#)
  - Preceding, [94](#)
  - Quarterly, [95](#)
  - Semiannual, [95](#)
  - Unadjusted, [94](#)
  - Weekday, [95](#)
- Day counters, [99](#)
- DayCounter
  - QuantLib::DayCounter, [355](#)
- Debugging macros, [147](#)
- debugMacros
  - QL\_TRACE, [148](#)
  - QL\_TRACE\_DISABLE, [147](#)
  - QL\_TRACE\_ENABLE, [147](#)
  - QL\_TRACE\_ENTER\_FUNCTION, [148](#)
  - QL\_TRACE\_EXIT\_FUNCTION, [148](#)
  - QL\_TRACE\_LOCATION, [149](#)
  - QL\_TRACE\_ON, [148](#)
  - QL\_TRACE\_VARIABLE, [149](#)
- DEFINE\_SEQUENCE\_STAT\_CONST\_-METHOD\_DOUBLE
  - sequencestatistics.hpp, [1164](#)
- DEFINE\_SEQUENCE\_STAT\_CONST\_-METHOD\_VOID

- sequencestatistics.hpp, 1164
- Derived
  - QuantLib::ExchangeRate, 413
- Design patterns, 136
- diffusion
  - QuantLib::BlackScholesProcess, 229
  - QuantLib::EulerDiscretization, 403
- Direct
  - QuantLib::ExchangeRate, 413
- dirtyPrice
  - QuantLib::Bond, 244, 245
- discount
  - QuantLib::YieldTermStructure, 938
- DiscountCurve
  - yieldtermstructures, 138
- discountFactor
  - QuantLib::InterestRate, 560
- discountImpl
  - QuantLib::CompoundForward, 309
  - QuantLib::ForwardRateStructure, 460
  - QuantLib::ZeroYieldStructure, 946
- Down
  - QuantLib::Rounding, 788
- downsideDeviation
  - QuantLib::GenericRiskStatistics, 503
  - QuantLib::IncrementalStatistics, 544
- downsideVariance
  - QuantLib::GenericRiskStatistics, 502
  - QuantLib::IncrementalStatistics, 544
- drift
  - QuantLib::BlackScholesProcess, 229
  - QuantLib::EulerDiscretization, 403
- duration
  - QuantLib::Cashflows, 288
- equivalentRate
  - QuantLib::InterestRate, 561
- Error
  - QuantLib::Error, 400
- errorEstimate
  - QuantLib::GeneralStatistics, 498
  - QuantLib::IncrementalStatistics, 544
- errors.hpp
  - QL\_ASSERT, 1030
  - QL\_ENSURE, 1030
  - QL\_FAIL, 1030
  - QL\_REQUIRE, 1030
- Eurex
  - QuantLib::Germany, 508
- evaluationDate
  - QuantLib::Settings, 802
- EveryFourthMonth
  - datetime, 95
- evolve
  - QuantLib::LiborForwardModelProcess, 620
  - QuantLib::StochasticProcess, 840
  - QuantLib::StochasticProcess1D, 843
- Exchange
  - QuantLib::Italy, 582
  - QuantLib::UnitedKingdom, 913
  - QuantLib::UnitedStates, 916
- ExchangeRate
  - QuantLib::ExchangeRate, 413
- expectation
  - QuantLib::OrnsteinUhlenbeckProcess, 741
  - QuantLib::StochasticProcess, 840
  - QuantLib::StochasticProcess1D, 843
  - QuantLib::StochasticProcessArray, 847
- expectationValue
  - QuantLib::GeneralStatistics, 499
- expectedShortfall
  - QuantLib::GenericRiskStatistics, 503
- FDAmericanEngine
  - vanillaengines, 110
- FDDividendAmericanEngine
  - vanillaengines, 110
- FDDividendEuropeanEngine
  - vanillaengines, 111
- FDDividendShoutEngine
  - vanillaengines, 111
- FDShoutEngine
  - vanillaengines, 111
- fetchResults
  - QuantLib::ForwardVanillaOption, 465
  - QuantLib::Instrument, 556
  - QuantLib::MultiAssetOption, 690
  - QuantLib::OneAssetOption, 725
  - QuantLib::OneAssetStrikedOption, 729
  - QuantLib::QuantoVanillaOption, 777
  - QuantLib::VanillaSwap, 926
- Financial instruments, 122
- findiff
  - BSMTermOperator, 118
- findIndex
  - QuantLib::TimeGrid, 884
- Finite-differences framework, 112
- FirstDerivative
  - QuantLib::CubicSpline, 337
- firstDerivativeAtCenter
  - QuantLib::SampledCurve, 792
- FixedCouponBond
  - QuantLib::FixedCouponBond, 439
- FixedCouponBondHelper
  - QuantLib::FixedCouponBondHelper, 441
- fixing
  - QuantLib::Index, 546

- QuantLib::Xibor, [935](#)
- FloatingRateBond
  - QuantLib::FloatingRateBond, [449](#)
- Floor
  - QuantLib::Rounding, [788](#)
- Following
  - datetime, [94](#)
- format
  - QuantLib::Currency, [345](#)
- formula
  - QuantLib::BlackModel, [225](#)
- Forward option engines, [106](#)
- ForwardFlatInterpolation
  - QuantLib::ForwardFlatInterpolation, [456](#)
- forwardRate
  - QuantLib::YieldTermStructure, [938](#)
- FrankfurtStockExchange
  - QuantLib::Germany, [508](#)
- freeze
  - QuantLib::LazyObject, [604](#)
- Frequency
  - datetime, [95](#)
- gaussianDownsideDeviation
  - QuantLib::GaussianStatistics, [487](#)
- gaussianDownsideVariance
  - QuantLib::GaussianStatistics, [487](#)
- gaussianExpectedShortfall
  - QuantLib::GaussianStatistics, [488](#)
- gaussianPercentile
  - QuantLib::GaussianStatistics, [488](#)
- gaussianPotentialUpside
  - QuantLib::GaussianStatistics, [488](#)
- gaussianRegret
  - QuantLib::GaussianStatistics, [488](#)
- gaussianTopPercentile
  - QuantLib::GaussianStatistics, [488](#)
- gaussianValueAtRisk
  - QuantLib::GaussianStatistics, [488](#)
- Generic macros, [142](#)
- Handle
  - QuantLib::Handle, [512](#)
- hasOccurred
  - QuantLib::Event, [410](#)
- History
  - QuantLib::History, [518](#), [519](#)
- HKEx
  - QuantLib::HongKong, [524](#)
- ICEX
  - QuantLib::Iceland, [532](#)
- impliedRate
  - QuantLib::InterestRate, [561](#)
- impliedVolatility
  - QuantLib::OneAssetOption, [725](#)
  - QuantLib::SingleAssetOption, [819](#)
- irr
  - QuantLib::Cashflows, [288](#)
- isBusinessDay
  - QuantLib::Calendar, [263](#)
- isEndOfMonth
  - QuantLib::Calendar, [263](#)
- isHoliday
  - QuantLib::Calendar, [263](#)
- isOnTime
  - QuantLib::DiscretizedAsset, [372](#)
- Iterator support, [145](#)
- iteratorMacros
  - QL\_FULL\_ITERATOR\_SUPPORT, [145](#)
- itmAssetProbability
  - QuantLib::BlackFormula, [222](#)
- itmCashProbability
  - QuantLib::BlackFormula, [222](#)
- itmProbability
  - QuantLib::BlackModel, [226](#)
- KnuthUniformRng
  - QuantLib::KnuthUniformRng, [594](#)
- KRX
  - QuantLib::SouthKorea, [832](#)
- kurtosis
  - QuantLib::GeneralStatistics, [498](#)
  - QuantLib::IncrementalStatistics, [545](#)
- Lagrange
  - QuantLib::CubicSpline, [337](#)
- latestDate
  - QuantLib::DepositRateHelper, [360](#)
  - QuantLib::FixedCouponBondHelper, [441](#)
  - QuantLib::FraRateHelper, [469](#)
  - QuantLib::FuturesRateHelper, [471](#)
  - QuantLib::RateHelper, [783](#)
  - QuantLib::SwapRateHelper, [856](#)
- Lattice methods, [125](#)
- LecuyerUniformRng
  - QuantLib::LecuyerUniformRng, [607](#)
- LevenbergMarquardt
  - QuantLib::LevenbergMarquardt, [609](#)
- limitMacros
  - QL\_EPSILON, [143](#)
  - QL\_MAX\_INTEGER, [143](#)
  - QL\_MAX\_REAL, [143](#)
  - QL\_MIN\_INTEGER, [143](#)
  - QL\_MIN\_POSITIVE\_REAL, [143](#)
  - QL\_MIN\_REAL, [143](#)
- LinearInterpolation
  - QuantLib::LinearInterpolation, [622](#)

- Link
  - QuantLib::Link, 625
- linkTo
  - QuantLib::Handle, 512
  - QuantLib::Link, 626
- localVolImpl
  - QuantLib::LocalVolCurve, 633
- LocalVolTermStructure
  - QuantLib::LocalVolTermStructure, 637
- LogLinearInterpolation
  - QuantLib::LogLinearInterpolation, 639
- lookup
  - QuantLib::ExchangeRateManager, 414
- mandatoryTimes
  - QuantLib::DiscretizedAsset, 372
  - QuantLib::DiscretizedDiscountBond, 374
  - QuantLib::DiscretizedOption, 375
- Market
  - QuantLib::Argentina, 174
  - QuantLib::CzechRepublic, 348
  - QuantLib::Germany, 508
  - QuantLib::HongKong, 524
  - QuantLib::Iceland, 532
  - QuantLib::India, 551
  - QuantLib::Indonesia, 553
  - QuantLib::Italy, 582
  - QuantLib::Mexico, 679
  - QuantLib::Singapore, 817
  - QuantLib::Slovakia, 825
  - QuantLib::SouthKorea, 832
  - QuantLib::Taiwan, 872
  - QuantLib::Ukraine, 911
  - QuantLib::UnitedKingdom, 913
  - QuantLib::UnitedStates, 916
- Math tools, 128
- max
  - QuantLib::GeneralStatistics, 499
  - QuantLib::IncrementalStatistics, 545
- mean
  - QuantLib::GeneralStatistics, 498
  - QuantLib::IncrementalStatistics, 544
- MersenneTwisterUniformRng
  - QuantLib::MersenneTwisterUniformRng, 675
- Merval
  - QuantLib::Argentina, 174
- min
  - QuantLib::GeneralStatistics, 498
  - QuantLib::IncrementalStatistics, 545
- miscMacros
  - QL\_DUMMY\_RETURN, 142
  - QL\_IO\_INIT, 142
- ModifiedFollowing
  - datetime, 95
- ModifiedPreceding
  - datetime, 94
- MonotonicCubicSpline
  - QuantLib::MonotonicCubicSpline, 684
- Monte Carlo framework, 130
- MonthEndReference
  - datetime, 95
- Monthly
  - datetime, 95
- name
  - QuantLib::Calendar, 263
  - QuantLib::DayCounter, 355
  - QuantLib::Index, 546
  - QuantLib::Xibor, 935
- NaturalCubicSpline
  - QuantLib::NaturalCubicSpline, 698
- NaturalMonotonicCubicSpline
  - QuantLib::NaturalMonotonicCubicSpline, 699
- next
  - QuantLib::KnuthUniformRng, 594
  - QuantLib::LecuyerUniformRng, 607
  - QuantLib::MersenneTwisterUniformRng, 675
- nextIMMdate
  - QuantLib::Date, 352
- nextRandomizer
  - QuantLib::RandomizedLDS, 780
- nextWeekday
  - QuantLib::Date, 352
- NoConversion
  - QuantLib::Money, 683
- NoFrequency
  - datetime, 95
- None
  - QuantLib::Rounding, 788
- NotAKnot
  - QuantLib::CubicSpline, 337
- notifyObservers
  - QuantLib::Observable, 720
- npv
  - QuantLib::Cashflows, 287, 288
- NSE
  - QuantLib::India, 551
- nthWeekday
  - QuantLib::Date, 352
- Numeric limits, 143
- Numeric types, 87
- NYSE
  - QuantLib::UnitedStates, 916
- Once



- datetime, 95
- operator+=
  - QuantLib::Matrix, 649
- operator==
  - QuantLib::Calendar, 264
  - QuantLib::DayCounter, 355
- Output manipulators, 146
- parRate
  - QuantLib::YieldTermStructure, 938
- partialRollback
  - QuantLib::Lattice, 599
  - QuantLib::NumericalMethod, 716
  - QuantLib::TsiveriotisFernandesLattice, 901
- percentile
  - QuantLib::GeneralStatistics, 499
- performCalculations
  - QuantLib::Bond, 245
  - QuantLib::Instrument, 557
  - QuantLib::LazyObject, 604
  - QuantLib::Stock, 848
  - QuantLib::Swap, 854
- Periodic
  - QuantLib::CubicSpline, 337
- postAdjustValues
  - QuantLib::DiscretizedAsset, 372
- postAdjustValuesImpl
  - QuantLib::DiscretizedAsset, 373
  - QuantLib::DiscretizedOption, 376
- potentialUpside
  - QuantLib::GenericRiskStatistics, 503
- preAdjustValues
  - QuantLib::DiscretizedAsset, 372
- preAdjustValuesImpl
  - QuantLib::DiscretizedAsset, 372
- Preceding
  - datetime, 94
- Pricing engines, 100
- PSE
  - QuantLib::CzechRepublic, 348
- pseudoSqrt
  - QuantLib::Matrix, 649
- ql/argsandresults.hpp, 949
- ql/calendar.hpp, 951
- ql/Calendars/argentina.hpp, 953
- ql/Calendars/beijing.hpp, 954
- ql/Calendars/bombay.hpp, 955
- ql/Calendars/bratislava.hpp, 956
- ql/Calendars/brazil.hpp, 957
- ql/Calendars/budapest.hpp, 958
- ql/Calendars/copenhagen.hpp, 959
- ql/Calendars/germany.hpp, 960
- ql/Calendars/helsinki.hpp, 961
- ql/Calendars/hongkong.hpp, 962
- ql/Calendars/iceland.hpp, 963
- ql/Calendars/indonesia.hpp, 964
- ql/Calendars/istanbul.hpp, 965
- ql/Calendars/italy.hpp, 966
- ql/Calendars/johannesburg.hpp, 967
- ql/Calendars/jointcalendar.hpp, 968
- ql/Calendars/mexico.hpp, 969
- ql/Calendars/nullcalendar.hpp, 970
- ql/Calendars/oslo.hpp, 971
- ql/Calendars/prague.hpp, 972
- ql/Calendars/riyadh.hpp, 973
- ql/Calendars/seoul.hpp, 974
- ql/Calendars/singapore.hpp, 975
- ql/Calendars/stockholm.hpp, 976
- ql/Calendars/sydney.hpp, 977
- ql/Calendars/taipei.hpp, 978
- ql/Calendars/taiwan.hpp, 979
- ql/Calendars/target.hpp, 980
- ql/Calendars/tokyo.hpp, 981
- ql/Calendars/toronto.hpp, 982
- ql/Calendars/ukraine.hpp, 983
- ql/Calendars/unitedkingdom.hpp, 984
- ql/Calendars/unitedstates.hpp, 985
- ql/Calendars/warsaw.hpp, 986
- ql/Calendars/wellington.hpp, 987
- ql/Calendars/zurich.hpp, 988
- ql/capvolstructures.hpp, 989
- ql/cashflow.hpp, 990
- ql/CashFlows/analysis.hpp, 991
- ql/CashFlows/basispointsensitivity.hpp, 992
- ql/CashFlows/cashflowvectors.hpp, 993
- ql/CashFlows/coupon.hpp, 994
- ql/CashFlows/dividend.hpp, 995
- ql/CashFlows/fixedratecoupon.hpp, 996
- ql/CashFlows/floatingratecoupon.hpp, 997
- ql/CashFlows/inarrearinindexedcoupon.hpp, 998
- ql/CashFlows/indexedcashflowvectors.hpp, 999
- ql/CashFlows/indexedcoupon.hpp, 1000
- ql/CashFlows/parcoupon.hpp, 1001
- ql/CashFlows/shortfloatingcoupon.hpp, 1002
- ql/CashFlows/shortindexedcoupon.hpp, 1003
- ql/CashFlows/simplecashflow.hpp, 1004
- ql/CashFlows/timebasket.hpp, 1005
- ql/CashFlows/upfrontindexedcoupon.hpp, 1006
- ql/Currencies/africa.hpp, 1007
- ql/Currencies/america.hpp, 1008
- ql/Currencies/asia.hpp, 1010
- ql/Currencies/europe.hpp, 1012
- ql/Currencies/exchangeratemanager.hpp, 1015
- ql/Currencies/oceania.hpp, 1016

- ql/currency.hpp, 1017
- ql/date.hpp, 1018
- ql/daycounter.hpp, 1021
- ql/DayCounters/actual360.hpp, 1022
- ql/DayCounters/actual365fixed.hpp, 1023
- ql/DayCounters/actualactual.hpp, 1024
- ql/DayCounters/one.hpp, 1025
- ql/DayCounters/simpliedaycounter.hpp, 1026
- ql/DayCounters/thirty360.hpp, 1027
- ql/discretizedasset.hpp, 1028
- ql/errors.hpp, 1029
- ql/event.hpp, 1032
- ql/exchangerate.hpp, 1033
- ql/exercise.hpp, 1034
- ql/FiniteDifferences/americancondition.hpp, 1035
- ql/FiniteDifferences/boundarycondition.hpp, 1036
- ql/FiniteDifferences/bsmoperator.hpp, 1037
- ql/FiniteDifferences/bsmtermoperator.hpp, 1038
- ql/FiniteDifferences/cranknicolson.hpp, 1039
- ql/FiniteDifferences/dminus.hpp, 1040
- ql/FiniteDifferences/dplus.hpp, 1041
- ql/FiniteDifferences/dplusdminus.hpp, 1042
- ql/FiniteDifferences/dzero.hpp, 1043
- ql/FiniteDifferences/expliciteuler.hpp, 1044
- ql/FiniteDifferences/fdtypedefs.hpp, 1045
- ql/FiniteDifferences/finitedifferencemodel.hpp, 1046
- ql/FiniteDifferences/impliciteuler.hpp, 1047
- ql/FiniteDifferences/mixedscheme.hpp, 1048
- ql/FiniteDifferences/onefactoroperator.hpp, 1049
- ql/FiniteDifferences/operatorfactory.hpp, 1050
- ql/FiniteDifferences/operatortraits.hpp, 1051
- ql/FiniteDifferences/parallelevolver.hpp, 1052
- ql/FiniteDifferences/pde.hpp, 1053
- ql/FiniteDifferences/pdebsm.hpp, 1054
- ql/FiniteDifferences/pdeshortrate.hpp, 1055
- ql/FiniteDifferences/shoutcondition.hpp, 1056
- ql/FiniteDifferences/stepcondition.hpp, 1057
- ql/FiniteDifferences/tridiagonaloperator.hpp, 1058
- ql/FiniteDifferences/valueatcenter.hpp, 1059
- ql/FiniteDifferences/zerocondition.hpp, 1060
- ql/grid.hpp, 1061
- ql/handle.hpp, 1062
- ql/history.hpp, 1063
- ql/index.hpp, 1064
- ql/Indexes/audlibor.hpp, 1065
- ql/Indexes/cadlibor.hpp, 1066
- ql/Indexes/cdor.hpp, 1067
- ql/Indexes/chflibor.hpp, 1068
- ql/Indexes/dkklibor.hpp, 1069
- ql/Indexes/euribor.hpp, 1070
- ql/Indexes/eurlibor.hpp, 1071
- ql/Indexes/gbp-libor.hpp, 1072
- ql/Indexes/indexmanager.hpp, 1073
- ql/Indexes/jibar.hpp, 1074
- ql/Indexes/jpylibor.hpp, 1075
- ql/Indexes/libor.hpp, 1076
- ql/Indexes/nzdlibor.hpp, 1077
- ql/Indexes/tibor.hpp, 1078
- ql/Indexes/trlibor.hpp, 1079
- ql/Indexes/usdlibor.hpp, 1080
- ql/Indexes/xibor.hpp, 1081
- ql/Indexes/zibor.hpp, 1082
- ql/instrument.hpp, 1083
- ql/Instruments/asianoption.hpp, 1084
- ql/Instruments/barrieroption.hpp, 1085
- ql/Instruments/basketoption.hpp, 1086
- ql/Instruments/bond.hpp, 1087
- ql/Instruments/callabilityschedule.hpp, 1088
- ql/Instruments/capfloor.hpp, 1089
- ql/Instruments/cliquetoption.hpp, 1090
- ql/Instruments/convertiblebond.hpp, 1091
- ql/Instruments/dividendschedule.hpp, 1093
- ql/Instruments/dividendvanillaoption.hpp, 1094
- ql/Instruments/europeanoption.hpp, 1095
- ql/Instruments/fixedcouponbond.hpp, 1096
- ql/Instruments/floatingratebond.hpp, 1097
- ql/Instruments/forwardvanillaoption.hpp, 1098
- ql/Instruments/multiassetoption.hpp, 1099
- ql/Instruments/oneassetoption.hpp, 1100
- ql/Instruments/oneassetstrikedoption.hpp, 1101
- ql/Instruments/payoffs.hpp, 1102
- ql/Instruments/quantoforwardvanillaoption.hpp, 1103
- ql/Instruments/quantovanillaoption.hpp, 1104
- ql/Instruments/simpleswap.hpp, 1105
- ql/Instruments/stock.hpp, 1106
- ql/Instruments/swap.hpp, 1107
- ql/Instruments/swaption.hpp, 1108
- ql/Instruments/vanillaoption.hpp, 1109
- ql/Instruments/zerocouponbond.hpp, 1110
- ql/interestrate.hpp, 1111
- ql/Lattices/binomialtree.hpp, 1112
- ql/Lattices/bsmlattice.hpp, 1114
- ql/Lattices/lattice.hpp, 1115
- ql/Lattices/lattice1d.hpp, 1116
- ql/Lattices/lattice2d.hpp, 1117
- ql/Lattices/tflattice.hpp, 1118
- ql/Lattices/tree.hpp, 1119
- ql/Lattices/trinomialtree.hpp, 1120



- ql/Math/array.hpp, 1121
- ql/Math/backwardflatinterpolation.hpp, 1122
- ql/Math/beta.hpp, 1123
- ql/Math/bicubicsplineinterpolation.hpp, 1124
- ql/Math/bilinearinterpolation.hpp, 1125
- ql/Math/binomialdistribution.hpp, 1126
- ql/Math/bivariatenormaldistribution.hpp, 1127
- ql/Math/chisquaredistribution.hpp, 1128
- ql/Math/choleskydecomposition.hpp, 1129
- ql/Math/comparison.hpp, 1130
- ql/Math/convergencestatistics.hpp, 1131
- ql/Math/cubicspline.hpp, 1132
- ql/Math/ discrepencystatistics.hpp, 1133
- ql/Math/errorfunction.hpp, 1134
- ql/Math/extrapolation.hpp, 1135
- ql/Math/factorial.hpp, 1136
- ql/Math/forwardflatinterpolation.hpp, 1137
- ql/Math/functional.hpp, 1138
- ql/Math/gammadistribution.hpp, 1139
- ql/Math/gaussianorthogonalpolynomial.hpp, 1140
- ql/Math/gaussianquadratures.hpp, 1141
- ql/Math/gaussianstatistics.hpp, 1143
- ql/Math/generalstatistics.hpp, 1144
- ql/Math/incompletestgamma.hpp, 1145
- ql/Math/incrementalstatistics.hpp, 1146
- ql/Math/interpolation.hpp, 1147
- ql/Math/interpolation2D.hpp, 1148
- ql/Math/kronrodintegral.hpp, 1149
- ql/Math/lexicographicalview.hpp, 1150
- ql/Math/linearinterpolation.hpp, 1151
- ql/Math/loglinearinterpolation.hpp, 1152
- ql/Math/matrix.hpp, 1153
- ql/Math/multicubicspline.hpp, 1154
- ql/Math/normaldistribution.hpp, 1156
- ql/Math/poissondistribution.hpp, 1157
- ql/Math/primenumbers.hpp, 1158
- ql/Math/pseudosqrt.hpp, 1159
- ql/Math/riskstatistics.hpp, 1160
- ql/Math/rounding.hpp, 1161
- ql/Math/sampledcurve.hpp, 1162
- ql/Math/segmentintegral.hpp, 1163
- ql/Math/sequencestatistics.hpp, 1164
- ql/Math/simpsonintegral.hpp, 1166
- ql/Math/statistics.hpp, 1167
- ql/Math/svd.hpp, 1168
- ql/Math/symmetriceigenvalues.hpp, 1169
- ql/Math/symmetricchurdecomposition.hpp, 1170
- ql/Math/tqreigendecomposition.hpp, 1171
- ql/Math/transformedgrid.hpp, 1172
- ql/Math/trapezoidintegral.hpp, 1173
- ql/money.hpp, 1174
- ql/MonteCarlo/brownianbridge.hpp, 1175
- ql/MonteCarlo/getcovariance.hpp, 1176
- ql/MonteCarlo/mctrails.hpp, 1177
- ql/MonteCarlo/mctypedefs.hpp, 1178
- ql/MonteCarlo/montecarlomodel.hpp, 1179
- ql/MonteCarlo/multipath.hpp, 1180
- ql/MonteCarlo/multipathgenerator.hpp, 1181
- ql/MonteCarlo/path.hpp, 1182
- ql/MonteCarlo/pathgenerator.hpp, 1183
- ql/MonteCarlo/pathpricer.hpp, 1184
- ql/MonteCarlo/sample.hpp, 1185
- ql/numericalmethod.hpp, 1186
- ql/Optimization/armijo.hpp, 1187
- ql/Optimization/conjugategradient.hpp, 1188
- ql/Optimization/constraint.hpp, 1189
- ql/Optimization/costfunction.hpp, 1190
- ql/Optimization/criteria.hpp, 1191
- ql/Optimization/leastsquare.hpp, 1192
- ql/Optimization/levenbergmarquardt.hpp, 1193
- ql/Optimization/linearssearch.hpp, 1194
- ql/Optimization/lmdif.hpp, 1195
- ql/Optimization/method.hpp, 1196
- ql/Optimization/problem.hpp, 1197
- ql/Optimization/simplex.hpp, 1198
- ql/Optimization/steepestdescent.hpp, 1199
- ql/option.hpp, 1200
- ql/Patterns/bridge.hpp, 1201
- ql/Patterns/composite.hpp, 1202
- ql/Patterns/curiouslyrecurring.hpp, 1203
- ql/Patterns/lazyobject.hpp, 1204
- ql/Patterns/observable.hpp, 1205
- ql/Patterns/singleton.hpp, 1206
- ql/Patterns/visitor.hpp, 1207
- ql/payoff.hpp, 1208
- ql/Pricers/discretegeometricaso.hpp, 1209
- ql/Pricers/mccliquestoption.hpp, 1210
- ql/Pricers/mcdiscretearithmeticsaso.hpp, 1211
- ql/Pricers/mceverest.hpp, 1212
- ql/Pricers/mchimalaya.hpp, 1213
- ql/Pricers/mcmaxbasket.hpp, 1214
- ql/Pricers/mcpagoda.hpp, 1215
- ql/Pricers/mcperformanceoption.hpp, 1216
- ql/Pricers/mcpricer.hpp, 1217
- ql/Pricers/singleassetoption.hpp, 1218
- ql/pricingengine.hpp, 1219
- ql/PricingEngines/americanpayoffatexpiry.hpp, 1220
- ql/PricingEngines/americanpayoffathit.hpp, 1221
- ql/PricingEngines/Asian/analytic\_cont\_geom\_av\_price.hpp, 1222
- ql/PricingEngines/Asian/analytic\_discr\_geom\_av\_price.hpp, 1223

- [ql/PricingEngines/Asian/mc\\_discr\\_arith\\_av\\_price.hpp, 1224](#)  
[ql/PricingEngines/Asian/mc\\_discr\\_geom\\_av\\_price.hpp, 1225](#)  
[ql/PricingEngines/Asian/mcdiscreteasianengine.hpp, 1226](#)  
[ql/PricingEngines/Barrier/analyticbarrierengine.hpp, 1227](#)  
[ql/PricingEngines/Barrier/mcbarrierengine.hpp, 1228](#)  
[ql/PricingEngines/Basket/mcamericanbasketengine.hpp, 1229](#)  
[ql/PricingEngines/Basket/mcbasketengine.hpp, 1230](#)  
[ql/PricingEngines/Basket/stulzengine.hpp, 1231](#)  
[ql/PricingEngines/blackformula.hpp, 1232](#)  
[ql/PricingEngines/blackmodel.hpp, 1233](#)  
[ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp, 1234](#)  
[ql/PricingEngines/CapFloor/blackcapfloorengine.hpp, 1235](#)  
[ql/PricingEngines/CapFloor/discretizedcapfloor.hpp, 1236](#)  
[ql/PricingEngines/CapFloor/treecapfloorengine.hpp, 1237](#)  
[ql/PricingEngines/Cliquet/analyticcliquetengine.hpp, 1238](#)  
[ql/PricingEngines/Cliquet/analyticperformanceengine.hpp, 1239](#)  
[ql/PricingEngines/Forward/forwardengine.hpp, 1240](#)  
[ql/PricingEngines/Forward/forwardperformanceengine.hpp, 1241](#)  
[ql/PricingEngines/genericmodelengine.hpp, 1242](#)  
[ql/PricingEngines/greeks.hpp, 1243](#)  
[ql/PricingEngines/Hybrid/binomialconvertibleengine.hpp, 1244](#)  
[ql/PricingEngines/Hybrid/discretizedconvertible.hpp, 1245](#)  
[ql/PricingEngines/latticeshortratemodelengine.hpp, 1246](#)  
[ql/PricingEngines/mcsimulation.hpp, 1247](#)  
[ql/PricingEngines/Quanto/quantoengine.hpp, 1248](#)  
[ql/PricingEngines/Swapption/blackswapptionengine.hpp, 1249](#)  
[ql/PricingEngines/Swapption/discretizedswapptionengine.hpp, 1250](#)  
[ql/PricingEngines/Swapption/g2swapptionengine.hpp, 1251](#)  
[ql/PricingEngines/Swapption/jamshidianswapptionengine.hpp, 1252](#)  
[ql/PricingEngines/Swapption/lfmswapptionengine.hpp, 1253](#)  
[ql/PricingEngines/Swapption/treeswapptionengine.hpp, 1254](#)  
[ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp, 1255](#)  
[ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp, 1256](#)  
[ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp, 1257](#)  
[ql/PricingEngines/Vanilla/analytichestonengine.hpp, 1258](#)  
[ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp, 1259](#)  
[ql/PricingEngines/Vanilla/batesengine.hpp, 1260](#)  
[ql/PricingEngines/Vanilla/binomialengine.hpp, 1261](#)  
[ql/PricingEngines/Vanilla/bjerkhundstenslandengine.hpp, 1262](#)  
[ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp, 1263](#)  
[ql/PricingEngines/Vanilla/fdamericanengine.hpp, 1264](#)  
[ql/PricingEngines/Vanilla/fdbermudanengine.hpp, 1265](#)  
[ql/PricingEngines/Vanilla/fdconditions.hpp, 1266](#)  
[ql/PricingEngines/Vanilla/fddividendamericanengine.hpp, 1267](#)  
[ql/PricingEngines/Vanilla/fddividendengine.hpp, 1268](#)  
[ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp, 1269](#)  
[ql/PricingEngines/Vanilla/fddividendshoutengine.hpp, 1270](#)  
[ql/PricingEngines/Vanilla/fdeuropeanengine.hpp, 1271](#)  
[ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp, 1272](#)  
[ql/PricingEngines/Vanilla/fdshoutengine.hpp, 1273](#)  
[ql/PricingEngines/Vanilla/fdstepconditionengine.hpp, 1274](#)  
[ql/PricingEngines/Vanilla/fdvanillaengine.hpp, 1275](#)  
[ql/PricingEngines/Vanilla/integralengine.hpp, 1276](#)  
[ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp, 1277](#)  
[ql/PricingEngines/Vanilla/juquadraticengine.hpp, 1278](#)  
[ql/PricingEngines/Vanilla/mcdigitalengine.hpp, 1279](#)

- ql/PricingEngines/Vanilla/mceuropeanengine.hpp, 1280
- ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp, 1281
- ql/PricingEngines/Vanilla/mcvanillaengine.hpp, 1282
- ql/Processes/blackscholesprocess.hpp, 1283
- ql/Processes/eulerdiscretization.hpp, 1284
- ql/Processes/geometricbrownianprocess.hpp, 1285
- ql/Processes/hestonprocess.hpp, 1286
- ql/Processes/lfmcovarParams.hpp, 1287
- ql/Processes/lfmhullwhiteparam.hpp, 1288
- ql/Processes/lfmprocess.hpp, 1289
- ql/Processes/merton76process.hpp, 1290
- ql/Processes/ornsteinuhlenbeckprocess.hpp, 1291
- ql/Processes/squarerootprocess.hpp, 1292
- ql/Processes/stochasticprocessarray.hpp, 1293
- ql/qldefines.hpp, 1294
- ql/quote.hpp, 1296
- ql/RandomNumbers/boxmullergaussianrng.hpp, 1297
- ql/RandomNumbers/centrallimitgaussianrng.hpp, 1298
- ql/RandomNumbers/faurersg.hpp, 1299
- ql/RandomNumbers/haltonrsg.hpp, 1300
- ql/RandomNumbers/inversecumulativerng.hpp, 1301
- ql/RandomNumbers/inversecumulativersg.hpp, 1302
- ql/RandomNumbers/knuthuniformrng.hpp, 1303
- ql/RandomNumbers/lecuyeruniformrng.hpp, 1304
- ql/RandomNumbers/mt19937uniformrng.hpp, 1305
- ql/RandomNumbers/randomizedlds.hpp, 1306
- ql/RandomNumbers/randomsequencegenerator.hpp, 1307
- ql/RandomNumbers/rngtraits.hpp, 1308
- ql/RandomNumbers/seedgenerator.hpp, 1310
- ql/RandomNumbers/sobolrsg.hpp, 1311
- ql/schedule.hpp, 1312
- ql/settings.hpp, 1313
- ql/ShortRateModels/calibrationhelper.hpp, 1314
- ql/ShortRateModels/CalibrationHelpers/caphelper.hpp, 1315
- ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp, 1316
- ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp, 1317
- ql/ShortRateModels/LiborMarketModels/lfmcovarproxy.hpp, 1318
- ql/ShortRateModels/LiborMarketModels/liborforwardmodel.hpp, 1319
- ql/ShortRateModels/LiborMarketModels/lmcorrmodel.hpp, 1320
- ql/ShortRateModels/LiborMarketModels/lmexpcorrmodel.hpp, 1321
- ql/ShortRateModels/LiborMarketModels/lmfixedvolmodel.hpp, 1322
- ql/ShortRateModels/LiborMarketModels/lmlinexpvolmodel.hpp, 1323
- ql/ShortRateModels/LiborMarketModels/lmvolmodel.hpp, 1324
- ql/ShortRateModels/model.hpp, 1325
- ql/ShortRateModels/onefactormodel.hpp, 1326
- ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp, 1327
- ql/ShortRateModels/OneFactorModels/coxingersollross.hpp, 1328
- ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp, 1329
- ql/ShortRateModels/OneFactorModels/hullwhite.hpp, 1330
- ql/ShortRateModels/OneFactorModels/vasicek.hpp, 1331
- ql/ShortRateModels/parameter.hpp, 1332
- ql/ShortRateModels/twofactormodel.hpp, 1333
- ql/ShortRateModels/TwoFactorModels/batesmodel.hpp, 1334
- ql/ShortRateModels/TwoFactorModels/g2.hpp, 1335
- ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp, 1336
- ql/solver1d.hpp, 1337
- ql/Solvers1D/bisection.hpp, 1338
- ql/Solvers1D/brent.hpp, 1339
- ql/Solvers1D/falseposition.hpp, 1340
- ql/Solvers1D/newton.hpp, 1341
- ql/Solvers1D/newtonsafe.hpp, 1342
- ql/Solvers1D/ridder.hpp, 1343
- ql/Solvers1D/secant.hpp, 1344
- ql/stochasticprocess.hpp, 1345
- ql/swaptionvolstructure.hpp, 1346
- ql/termstructure.hpp, 1347
- ql/TermStructures/affinetermstructure.hpp, 1348
- ql/TermStructures/bondhelpers.hpp, 1349
- ql/TermStructures/bootstraptraits.hpp, 1350
- ql/TermStructures/compoundforward.hpp, 1351
- ql/TermStructures/discountcurve.hpp, 1352
- ql/TermStructures/drifttermstructure.hpp, 1353

- ql/TermStructures/extendeddiscountcurve.hpp, 1354
- ql/TermStructures/flatforward.hpp, 1355
- ql/TermStructures/forwardcurve.hpp, 1356
- ql/TermStructures/forwardspreadedtermstructure.hpp, 1357
- ql/TermStructures/forwardstructure.hpp, 1358
- ql/TermStructures/IMPLIEDTERMSTRUCTURE.hpp, 1359
- ql/TermStructures/piecewiseflatforward.hpp, 1360
- ql/TermStructures/piecewiseyieldcurve.hpp, 1361
- ql/TermStructures/quantotermstructure.hpp, 1362
- ql/TermStructures/ratehelpers.hpp, 1363
- ql/TermStructures/zerocurve.hpp, 1364
- ql/TermStructures/zerospreadedtermstructure.hpp, 1365
- ql/TermStructures/zeroyieldstructure.hpp, 1366
- ql/timegrid.hpp, 1367
- ql/types.hpp, 1368
- ql/Utilities/dataformatters.hpp, 1370
- ql/Utilities/dataparsers.hpp, 1371
- ql/Utilities/disposable.hpp, 1372
- ql/Utilities/null.hpp, 1373
- ql/Utilities/observablevalue.hpp, 1374
- ql/Utilities/steppingiterator.hpp, 1375
- ql/Utilities/strings.hpp, 1376
- ql/Utilities/tracing.hpp, 1377
- ql/Volatilities/blackconstantvol.hpp, 1379
- ql/Volatilities/blackvariancecurve.hpp, 1380
- ql/Volatilities/blackvariancesurface.hpp, 1381
- ql/Volatilities/capflatvolvector.hpp, 1382
- ql/Volatilities/capletconstantvol.hpp, 1383
- ql/Volatilities/capletvariancecurve.hpp, 1384
- ql/Volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp, 1385
- ql/Volatilities/localconstantvol.hpp, 1386
- ql/Volatilities/localvolcurve.hpp, 1387
- ql/Volatilities/localvolsurface.hpp, 1388
- ql/Volatilities/swaptionvolmatrix.hpp, 1389
- ql/voltermstructure.hpp, 1390
- ql/yieldtermstructure.hpp, 1391
- QL\_ASSERT
  - errors.hpp, 1030
- QL\_DUMMY\_RETURN
  - miscMacros, 142
- QL\_ENSURE
  - errors.hpp, 1030
- QL\_EPSILON
  - limitMacros, 143
- QL\_FAIL
  - errors.hpp, 1030
- QL\_FULL\_ITERATOR\_SUPPORT
  - iteratorMacros, 145
- QL\_IO\_INIT
  - miscMacros, 142
- QL\_MAX\_INTEGER
  - limitMacros, 143
- QL\_MAX\_REAL
  - limitMacros, 143
- QL\_MIN\_INTEGER
  - limitMacros, 143
- QL\_MIN\_POSITIVE\_REAL
  - limitMacros, 143
- QL\_MIN\_REAL
  - limitMacros, 143
- QL\_REQUIRE
  - errors.hpp, 1030
- QL\_TRACE
  - debugMacros, 148
- QL\_TRACE\_DISABLE
  - debugMacros, 147
- QL\_TRACE\_ENABLE
  - debugMacros, 147
- QL\_TRACE\_ENTER\_FUNCTION
  - debugMacros, 148
- QL\_TRACE\_EXIT\_FUNCTION
  - debugMacros, 148
- QL\_TRACE\_LOCATION
  - debugMacros, 149
- QL\_TRACE\_ON
  - debugMacros, 148
- QL\_TRACE\_VARIABLE
  - debugMacros, 149
- QL\_TYPENAME
  - templateMacros, 144
- QuantLib macros, 141
- QuantLib::Actual360, 151
- QuantLib::Actual365Fixed, 152
- QuantLib::ActualActual, 153
- QuantLib::AcyclicVisitor, 154
- QuantLib::AdditiveEQPBinoMialTree, 155
- QuantLib::AffineModel, 156
- QuantLib::AffineTermStructure, 157
- QuantLib::AffineTermStructure
  - update, 158
- QuantLib::AmericanCondition, 159
- QuantLib::AmericanExercise, 160
- QuantLib::AmericanPayoffAtExpiry, 161
- QuantLib::AmericanPayoffAtHit, 162
- QuantLib::AnalyticBarrierEngine, 163
- QuantLib::AnalyticCapFloorEngine, 164
- QuantLib::AnalyticCliquetEngine, 165
- QuantLib::AnalyticContinuousGeometricAveragePriceAsianEngine, 166

- QuantLib::AnalyticDigitalAmericanEngine, 167
- QuantLib::AnalyticDiscreteGeometricAveragePriceBinomialEngine, 168
- QuantLib::AnalyticDividendEuropeanEngine, 169
- QuantLib::AnalyticEuropeanEngine, 170
- QuantLib::AnalyticHestonEngine, 171
- QuantLib::AnalyticPerformanceEngine, 172
- QuantLib::Argentina, 173
  - Merval, 174
- QuantLib::Argentina
  - Market, 174
- QuantLib::Arguments, 175
- QuantLib::ArmijoLineSearch, 176
- QuantLib::Array, 177
- QuantLib::ARSCurrency, 180
- QuantLib::AssetOrNothingPayoff, 181
- QuantLib::ATSCurrency, 182
- QuantLib::AUDCurrency, 183
- QuantLib::AUDLibor, 184
- QuantLib::Australia, 185
- QuantLib::Average, 186
- QuantLib::BackwardFlat, 187
- QuantLib::BackwardFlatInterpolation, 188
- QuantLib::BackwardFlatInterpolation
  - BackwardFlatInterpolation, 188
- QuantLib::BaroneAdesiWhaleyApproximationEngine, 189
- QuantLib::Barrier, 190
- QuantLib::BarrierOption, 191
- QuantLib::BarrierOption
  - setupArguments, 192
- QuantLib::BarrierOption::arguments, 193
- QuantLib::BarrierOption::engine, 194
- QuantLib::BasketOption, 195
- QuantLib::BasketOption
  - setupArguments, 196
- QuantLib::BasketOption::arguments, 197
- QuantLib::BasketOption::engine, 198
- QuantLib::BatesEngine, 199
- QuantLib::BatesModel, 201
- QuantLib::BDTCurrency, 202
- QuantLib::BEFCurrency, 203
- QuantLib::BermudanExercise, 204
- QuantLib::BGLCurrency, 205
- QuantLib::Bicubic, 206
- QuantLib::BicubicSpline, 207
- QuantLib::BicubicSpline
  - BicubicSpline, 207
- QuantLib::Bilinear, 208
- QuantLib::BilinearInterpolation, 209
- QuantLib::BilinearInterpolation
  - BilinearInterpolation, 209
- QuantLib::BinomialConvertibleEngine, 210
- QuantLib::BinomialDistribution, 211
- QuantLib::BinomialTree, 212
- QuantLib::BinomialVanillaEngine, 213
- QuantLib::Bisection, 214
- QuantLib::BivariateCumulativeNormalDistributionDr78, 215
- QuantLib::BivariateCumulativeNormalDistributionWe04DP, 216
- QuantLib::Bjerk SundStenslandApproximationEngine, 217
- QuantLib::BlackCapFloorEngine, 218
- QuantLib::BlackConstantVol, 219
- QuantLib::BlackFormula, 221
- QuantLib::BlackFormula
  - itmAssetProbability, 222
  - itmCashProbability, 222
- QuantLib::BlackKarasinski, 223
- QuantLib::BlackKarasinski::Dynamics, 224
- QuantLib::BlackModel, 225
- QuantLib::BlackModel
  - formula, 225
  - itmProbability, 226
  - update, 225
- QuantLib::BlackScholesLattice, 227
- QuantLib::BlackScholesProcess, 228
- QuantLib::BlackScholesProcess
  - apply, 229
  - diffusion, 229
  - drift, 229
  - time, 229
  - update, 229
- QuantLib::BlackSwaptionEngine, 230
- QuantLib::BlackVarianceCurve, 231
- QuantLib::BlackVarianceSurface, 233
- QuantLib::BlackVarianceTermStructure, 235
- QuantLib::BlackVarianceTermStructure
  - BlackVarianceTermStructure, 236
  - blackVolImpl, 236
- QuantLib::BlackVolatilityTermStructure, 237
- QuantLib::BlackVolatilityTermStructure
  - blackVarianceImpl, 238
  - BlackVolatilityTermStructure, 238
- QuantLib::BlackVolTermStructure, 239
- QuantLib::BlackVolTermStructure
  - BlackVolTermStructure, 241
- QuantLib::Bond, 242
- QuantLib::Bond
  - accruedAmount, 245
  - Bond, 244
  - businessDayConvention, 244
  - cashflows, 244
  - cleanPrice, 244
  - dirtyPrice, 244, 245



- performCalculations, 245
- yield, 244, 245
- QuantLib::BoundaryCondition, 246
- QuantLib::BoundaryCondition
  - applyAfterApplying, 246
  - applyAfterSolving, 247
  - applyBeforeApplying, 246
  - applyBeforeSolving, 246
  - setTime, 247
  - Side, 246
- QuantLib::BoundaryConstraint, 248
- QuantLib::BoxMullerGaussianRng, 249
- QuantLib::BPSBasketCalculator, 250
- QuantLib::Brazil, 251
- QuantLib::Brent, 252
- QuantLib::Bridge, 253
- QuantLib::BRLCurrency, 254
- QuantLib::BrownianBridge, 255
- QuantLib::BSMOperator, 256
- QuantLib::BYRCurrency, 257
- QuantLib::CADCurrency, 258
- QuantLib::CADLibor, 259
- QuantLib::Calendar, 260
- QuantLib::Calendar
  - addHoliday, 263
  - adjust, 264
  - advance, 264
  - Calendar, 263
  - isBusinessDay, 263
  - isEndOfMonth, 263
  - isHoliday, 263
  - name, 263
  - operator==, 264
  - removeHoliday, 263
- QuantLib::Calendar::OrthodoxImpl, 265
- QuantLib::Calendar::WesternImpl, 266
- QuantLib::CalendarImpl, 267
- QuantLib::CalibrationHelper, 268
- QuantLib::CalibrationHelper
  - update, 269
- QuantLib::Canada, 270
- QuantLib::Cap, 271
- QuantLib::CapFloor, 272
- QuantLib::CapFloor
  - setupArguments, 273
- QuantLib::CapFloor::arguments, 274
- QuantLib::CapFloor::results, 275
- QuantLib::CapHelper, 276
- QuantLib::CapletConstantVolatility, 277
- QuantLib::CapletVolatilityStructure, 279
- QuantLib::CapletVolatilityStructure
  - CapletVolatilityStructure, 280
- QuantLib::CapVolatilityStructure, 281
- QuantLib::CapVolatilityStructure
  - CapVolatilityStructure, 282
- QuantLib::CapVolatilityVector, 283
- QuantLib::CapVolatilityVector
  - update, 284
- QuantLib::CashFlow, 285
- QuantLib::CashFlow
  - amount, 285
  - date, 285
- QuantLib::Cashflows, 287
- QuantLib::Cashflows
  - bps, 288
  - convexity, 289
  - duration, 288
  - irr, 288
  - npv, 287, 288
- QuantLib::CashOrNothingPayoff, 290
- QuantLib::Cdor, 291
- QuantLib::CeilingTruncation, 292
- QuantLib::CHFCurrency, 293
- QuantLib::CHFLibor, 294
- QuantLib::China, 295
- QuantLib::CLGaussianRng, 296
- QuantLib::CliquetOption, 297
- QuantLib::CliquetOption
  - setupArguments, 298
- QuantLib::CliquetOption::arguments, 299
- QuantLib::CliquetOption::engine, 300
- QuantLib::ClosestRounding, 301
- QuantLib::CLPCurrency, 302
- QuantLib::CNYCurrency, 303
- QuantLib::Collar, 304
- QuantLib::Composite, 305
- QuantLib::CompositeConstraint, 306
- QuantLib::CompositeQuote, 307
- QuantLib::CompositeQuote
  - update, 307
- QuantLib::CompoundForward, 308
- QuantLib::CompoundForward
  - discountImpl, 309
  - zeroYieldImpl, 309
- QuantLib::ConjugateGradient, 310
- QuantLib::ConstantParameter, 311
- QuantLib::Constraint, 312
- QuantLib::ConstraintImpl, 313
- QuantLib::ContinuousAveragingAsianOption, 314
- QuantLib::ContinuousAveragingAsianOption
  - setupArguments, 315
- QuantLib::ContinuousAveragingAsianOption::arguments, 316
- QuantLib::ContinuousAveragingAsianOption::engine, 317
- QuantLib::ConvergenceStatistics, 318

- QuantLib::ConvertibleBond::option::arguments, 319
- QuantLib::ConvertibleBond::option::engine, 320
- QuantLib::ConvertibleFixedCouponBond, 321
- QuantLib::ConvertibleFloatingRateBond, 322
- QuantLib::ConvertibleZeroCouponBond, 323
- QuantLib::COPCurrency, 324
- QuantLib::CostFunction, 325
- QuantLib::Coupon, 326
- QuantLib::Coupon
  - Coupon, 327
  - date, 327
- QuantLib::CovarianceDecomposition, 328
- QuantLib::CovarianceDecomposition
  - correlationMatrix, 328
  - CovarianceDecomposition, 328
  - standardDeviations, 328
  - variances, 328
- QuantLib::CoxIngersollRoss, 329
- QuantLib::CoxIngersollRoss::Dynamics, 331
- QuantLib::CoxRossRubinstein, 332
- QuantLib::CrankNicolson, 333
- QuantLib::Cubic, 335
- QuantLib::CubicSpline, 336
  - FirstDerivative, 337
  - Lagrange, 337
  - NotAKnot, 337
  - Periodic, 337
  - SecondDerivative, 337
- QuantLib::CubicSpline
  - BoundaryCondition, 337
  - CubicSpline, 337
- QuantLib::CumulativeBinomialDistribution, 338
- QuantLib::CumulativeNormalDistribution, 339
- QuantLib::CumulativePoissonDistribution, 340
- QuantLib::CuriouslyRecurringTemplate, 341
- QuantLib::Currency, 342
- QuantLib::Currency
  - Currency, 345
  - format, 345
- QuantLib::CYPCurrency, 346
- QuantLib::CzechRepublic, 347
  - PSE, 348
- QuantLib::CzechRepublic
  - Market, 348
- QuantLib::CZKCurrency, 349
- QuantLib::Date, 350
- QuantLib::Date
  - nextIMMdate, 352
  - nextWeekday, 352
  - nthWeekday, 352
- QuantLib::DayCounter, 354
- QuantLib::DayCounter
  - DayCounter, 355
  - name, 355
  - operator==, 355
- QuantLib::DayCounterImpl, 356
- QuantLib::DEMCurrency, 357
- QuantLib::Denmark, 358
- QuantLib::DepositRateHelper, 359
- QuantLib::DepositRateHelper
  - latestDate, 360
  - setTermStructure, 359
- QuantLib::DerivedQuote, 361
- QuantLib::DerivedQuote
  - update, 361
- QuantLib::DirichletBC, 362
- QuantLib::DirichletBC
  - applyAfterApplying, 362
  - applyAfterSolving, 362
  - applyBeforeApplying, 362
  - applyBeforeSolving, 362
  - setTime, 363
- QuantLib::Discount, 364
- QuantLib::DiscrepancyStatistics, 365
- QuantLib::DiscreteAveragingAsianOption, 366
- QuantLib::DiscreteAveragingAsianOption
  - setupArguments, 367
- QuantLib::DiscreteAveragingAsianOption::arguments, 368
- QuantLib::DiscreteAveragingAsianOption::engine, 369
- QuantLib::DiscreteGeometricASO, 370
- QuantLib::DiscretizedAsset, 371
- QuantLib::DiscretizedAsset
  - adjustValues, 372
  - isOnTime, 372
  - mandatoryTimes, 372
  - postAdjustValues, 372
  - postAdjustValuesImpl, 373
  - preAdjustValues, 372
  - preAdjustValuesImpl, 372
  - reset, 372
- QuantLib::DiscretizedDiscountBond, 374
- QuantLib::DiscretizedDiscountBond
  - mandatoryTimes, 374
  - reset, 374
- QuantLib::DiscretizedOption, 375
- QuantLib::DiscretizedOption
  - mandatoryTimes, 375
  - postAdjustValuesImpl, 376
  - reset, 375
- QuantLib::Disposable, 377
- QuantLib::Dividend, 378

- QuantLib::Dividend
  - amount, [379](#)
  - date, [378](#)
- QuantLib::DividendVanillaOption, [380](#)
- QuantLib::DividendVanillaOption
  - setupArguments, [381](#)
- QuantLib::DividendVanillaOption::arguments, [382](#)
- QuantLib::DividendVanillaOption::engine, [383](#)
- QuantLib::DKKCurrency, [384](#)
- QuantLib::DKKLabor, [385](#)
- QuantLib::DMinus, [386](#)
- QuantLib::DownRounding, [387](#)
- QuantLib::DPlus, [388](#)
- QuantLib::DPlusDMinus, [389](#)
- QuantLib::DriftTermStructure, [390](#)
- QuantLib::Duration, [392](#)
- QuantLib::DZero, [393](#)
- QuantLib::EarlyExercise, [394](#)
- QuantLib::EEKCurrency, [395](#)
- QuantLib::EndCriteria, [396](#)
- QuantLib::EqualJumpsBinomialTree, [398](#)
- QuantLib::EqualProbabilitiesBinomialTree, [399](#)
- QuantLib::Error, [400](#)
- QuantLib::Error
  - ~Error, [400](#)
  - Error, [400](#)
- QuantLib::ErrorFunction, [401](#)
- QuantLib::ESPCurrency, [402](#)
- QuantLib::EulerDiscretization, [403](#)
- QuantLib::EulerDiscretization
  - covariance, [404](#)
  - diffusion, [403](#)
  - drift, [403](#)
  - variance, [404](#)
- QuantLib::EURCurrency, [405](#)
- QuantLib::Euribor, [406](#)
- QuantLib::EURLabor, [407](#)
- QuantLib::EuropeanExercise, [408](#)
- QuantLib::EuropeanOption, [409](#)
- QuantLib::Event, [410](#)
- QuantLib::Event
  - hasOccurred, [410](#)
- QuantLib::ExchangeRate, [412](#)
  - Derived, [413](#)
  - Direct, [413](#)
- QuantLib::ExchangeRate
  - ExchangeRate, [413](#)
  - Type, [413](#)
- QuantLib::ExchangeRateManager, [414](#)
- QuantLib::ExchangeRateManager
  - add, [414](#)
  - lookup, [414](#)
- QuantLib::Exercise, [416](#)
- QuantLib::ExplicitEuler, [417](#)
- QuantLib::ExtendedCoxIngersollRoss, [419](#)
- QuantLib::ExtendedCoxIngersollRoss::Dynamics, [421](#)
- QuantLib::ExtendedCoxIngersollRoss::FittingParameter, [422](#)
- QuantLib::ExtendedDiscountCurve, [423](#)
- QuantLib::ExtendedDiscountCurve
  - compoundForwardImpl, [424](#)
  - update, [424](#)
  - zeroYieldImpl, [424](#)
- QuantLib::Extrapolator, [425](#)
- QuantLib::Factorial, [426](#)
- QuantLib::FalsePosition, [427](#)
- QuantLib::FaureRsg, [428](#)
- QuantLib::FDAmericanCondition, [429](#)
- QuantLib::FDBermudanEngine, [430](#)
- QuantLib::FDDividendEngineMerton73, [431](#)
- QuantLib::FDDividendEngineShiftScale, [432](#)
- QuantLib::FDEuropeanEngine, [433](#)
- QuantLib::FDStepConditionEngine, [434](#)
- QuantLib::FIMCurrency, [435](#)
- QuantLib::FiniteDifferenceModel, [436](#)
- QuantLib::FiniteDifferenceModel
  - rollback, [436](#)
- QuantLib::Finland, [437](#)
- QuantLib::FixedCouponBond, [438](#)
- QuantLib::FixedCouponBond
  - FixedCouponBond, [439](#)
- QuantLib::FixedCouponBondHelper, [440](#)
- QuantLib::FixedCouponBondHelper
  - FixedCouponBondHelper, [441](#)
  - latestDate, [441](#)
  - setTermStructure, [441](#)
- QuantLib::FixedDividend, [442](#)
- QuantLib::FixedDividend
  - amount, [442](#)
- QuantLib::FixedRateCoupon, [444](#)
- QuantLib::FixedRateCoupon
  - amount, [445](#)
- QuantLib::FlatForward, [446](#)
- QuantLib::FlatForward
  - update, [447](#)
- QuantLib::FloatingRateBond, [448](#)
- QuantLib::FloatingRateBond
  - FloatingRateBond, [449](#)
- QuantLib::FloatingRateCoupon, [450](#)
- QuantLib::Floor, [452](#)
- QuantLib::FloorTruncation, [453](#)
- QuantLib::ForwardEngine, [454](#)
- QuantLib::ForwardFlat, [455](#)
- QuantLib::ForwardFlatInterpolation, [456](#)
- QuantLib::ForwardFlatInterpolation
  - ForwardFlatInterpolation, [456](#)



- QuantLib::ForwardOptionArguments, 457
- QuantLib::ForwardPerformanceEngine, 458
- QuantLib::ForwardRate, 459
- QuantLib::ForwardRateStructure, 460
- QuantLib::ForwardRateStructure
  - discountImpl, 460
  - zeroYieldImpl, 461
- QuantLib::ForwardSpreadedTermStructure, 462
- QuantLib::ForwardVanillaOption, 464
- QuantLib::ForwardVanillaOption
  - fetchResults, 465
  - setupArguments, 465
- QuantLib::FractionalDividend, 466
- QuantLib::FractionalDividend
  - amount, 466
- QuantLib::FraRateHelper, 468
- QuantLib::FraRateHelper
  - latestDate, 469
  - setTermStructure, 468
- QuantLib::FRFCurrency, 470
- QuantLib::FuturesRateHelper, 471
- QuantLib::FuturesRateHelper
  - latestDate, 471
- QuantLib::G2, 472
- QuantLib::G2::FittingParameter, 474
- QuantLib::G2SwaptionEngine, 475
- QuantLib::GammaFunction, 476
- QuantLib::GapPayoff, 477
- QuantLib::GaussChebyshev2thIntegration, 478
- QuantLib::GaussChebyshevIntegration, 479
- QuantLib::GaussGegenbauerIntegration, 480
- QuantLib::GaussHermiteIntegration, 481
- QuantLib::GaussHermitePolynomial, 482
- QuantLib::GaussHyperbolicIntegration, 483
- QuantLib::GaussHyperbolicPolynomial, 484
- QuantLib::GaussianOrthogonalPolynomial, 485
- QuantLib::GaussianQuadrature, 486
- QuantLib::GaussianStatistics, 487
- QuantLib::GaussianStatistics
  - gaussianDownsideDeviation, 487
  - gaussianDownsideVariance, 487
  - gaussianExpectedShortfall, 488
  - gaussianPercentile, 488
  - gaussianPotentialUpside, 488
  - gaussianRegret, 488
  - gaussianTopPercentile, 488
  - gaussianValueAtRisk, 488
- QuantLib::GaussJacobiIntegration, 490
- QuantLib::GaussJacobiPolynomial, 491
- QuantLib::GaussLaguerreIntegration, 492
- QuantLib::GaussLaguerrePolynomial, 493
- QuantLib::GaussLegendreIntegration, 494
- QuantLib::GBPCurrency, 495
- QuantLib::GBPLibor, 496
- QuantLib::GeneralStatistics, 497
- QuantLib::GeneralStatistics
  - add, 499
  - errorEstimate, 498
  - expectationValue, 499
  - kurtosis, 498
  - max, 499
  - mean, 498
  - min, 498
  - percentile, 499
  - skewness, 498
  - standardDeviation, 498
  - topPercentile, 499
  - variance, 498
- QuantLib::GenericEngine, 500
- QuantLib::GenericModelEngine, 501
- QuantLib::GenericModelEngine
  - update, 501
- QuantLib::GenericRiskStatistics, 502
- QuantLib::GenericRiskStatistics
  - averageShortfall, 504
  - downsideDeviation, 503
  - downsideVariance, 502
  - expectedShortfall, 503
  - potentialUpside, 503
  - regret, 503
  - semiDeviation, 502
  - semiVariance, 502
  - shortfall, 503
  - valueAtRisk, 503
- QuantLib::GeometricBrownianMotionProcess, 505
- QuantLib::Germany, 506
  - Eurex, 508
  - FrankfurtStockExchange, 508
  - Settlement, 508
  - Xetra, 508
- QuantLib::Germany
  - Market, 508
- QuantLib::GRDCurrency, 509
- QuantLib::Greeks, 510
- QuantLib::HaltonRsg, 511
- QuantLib::Handle, 512
- QuantLib::Handle
  - Handle, 512
  - linkTo, 512
- QuantLib::HestonModel, 513
- QuantLib::HestonModelHelper, 514
- QuantLib::HestonProcess, 515
- QuantLib::HestonProcess
  - apply, 516
  - time, 516

- QuantLib::History, 517
- QuantLib::History
  - History, 518, 519
- QuantLib::History::const\_iterator, 520
- QuantLib::History::Entry, 521
- QuantLib::HKDCurrency, 522
- QuantLib::HongKong, 523
  - HKEEx, 524
- QuantLib::HongKong
  - Market, 524
- QuantLib::HUFCurrency, 525
- QuantLib::HullWhite, 526
- QuantLib::HullWhite::Dynamics, 528
- QuantLib::HullWhite::FittingParameter, 529
- QuantLib::Hungary, 530
- QuantLib::Iceland, 531
  - ICEX, 532
- QuantLib::Iceland
  - Market, 532
- QuantLib::IEPCurrency, 533
- QuantLib::ILSCurrency, 534
- QuantLib::IMM, 535
- QuantLib::ImplicitEuler, 536
- QuantLib::ImpliedTermStructure, 537
- QuantLib::ImpliedVolTermStructure, 539
- QuantLib::InArrearIndexedCoupon, 541
- QuantLib::IncrementalStatistics, 543
- QuantLib::IncrementalStatistics
  - add, 545
  - addSequence, 545
  - downsideDeviation, 544
  - downsideVariance, 544
  - errorEstimate, 544
  - kurtosis, 545
  - max, 545
  - mean, 544
  - min, 545
  - skewness, 545
  - standardDeviation, 544
  - variance, 544
- QuantLib::Index, 546
- QuantLib::Index
  - fixing, 546
  - name, 546
- QuantLib::IndexedCoupon, 547
- QuantLib::IndexedCoupon
  - amount, 548
  - update, 548
- QuantLib::IndexManager, 549
- QuantLib::India, 550
  - NSE, 551
- QuantLib::India
  - Market, 551
- QuantLib::Indonesia, 552
  - BEJ, 553
- QuantLib::Indonesia
  - Market, 553
- QuantLib::INRCurrency, 554
- QuantLib::Instrument, 555
- QuantLib::Instrument
  - calculate, 556
  - fetchResults, 556
  - performCalculations, 557
  - setPricingEngine, 556
  - setupArguments, 556
  - setupExpired, 557
- QuantLib::IntegralEngine, 558
- QuantLib::InterestRate, 559
- QuantLib::InterestRate
  - compoundFactor, 560
  - discountFactor, 560
  - equivalentRate, 561
  - impliedRate, 561
- QuantLib::InterpolatedDiscountCurve, 562
- QuantLib::InterpolatedForwardCurve, 564
- QuantLib::InterpolatedForwardCurve
  - zeroYieldImpl, 565
- QuantLib::InterpolatedZeroCurve, 566
- QuantLib::Interpolation, 568
- QuantLib::Interpolation2D, 569
- QuantLib::Interpolation2D::templateImpl, 570
- QuantLib::Interpolation2DImpl, 571
- QuantLib::Interpolation::templateImpl, 572
- QuantLib::InterpolationImpl, 573
- QuantLib::InverseCumulativeNormal, 574
- QuantLib::InverseCumulativePoisson, 575
- QuantLib::InverseCumulativeRng, 576
- QuantLib::InverseCumulativeRsg, 577
- QuantLib::IQDCurrency, 578
- QuantLib::IRRCurrency, 579
- QuantLib::ISKCurrency, 580
- QuantLib::Italy, 581
  - Exchange, 582
  - Settlement, 582
- QuantLib::Italy
  - Market, 582
- QuantLib::ITLCurrency, 583
- QuantLib::JamshidianSwaptionEngine, 584
- QuantLib::Japan, 585
- QuantLib::JarrowRudd, 587
- QuantLib::Jibar, 588
- QuantLib::JointCalendar, 589
- QuantLib::JPYCurrency, 590
- QuantLib::JPYLibor, 591
- QuantLib::JumpDiffusionEngine, 592
- QuantLib::JuQuadraticApproximationEngine, 593
- QuantLib::KnuthUniformRng, 594

- QuantLib::KnuthUniformRng
  - KnuthUniformRng, 594
  - next, 594
- QuantLib::KronrodIntegral, 595
- QuantLib::KRWCurrency, 596
- QuantLib::KWDCurrency, 597
- QuantLib::Lattice, 598
- QuantLib::Lattice
  - partialRollback, 599
  - rollback, 599
- QuantLib::Lattice1D, 600
- QuantLib::Lattice2D, 601
- QuantLib::LatticeShortRateModelEngine, 602
- QuantLib::LatticeShortRateModelEngine
  - update, 602
- QuantLib::LazyObject, 603
- QuantLib::LazyObject
  - calculate, 604
  - freeze, 604
  - performCalculations, 604
  - recalculate, 604
  - unfreeze, 604
  - update, 603
- QuantLib::LeastSquareFunction, 605
- QuantLib::LeastSquareProblem, 606
- QuantLib::LeastSquareProblem
  - targetValueAndGradient, 606
- QuantLib::LecuyerUniformRng, 607
- QuantLib::LecuyerUniformRng
  - LecuyerUniformRng, 607
  - next, 607
- QuantLib::LeisenReimer, 608
- QuantLib::LevenbergMarquardt, 609
- QuantLib::LevenbergMarquardt
  - LevenbergMarquardt, 609
- QuantLib::LexicographicalView, 610
- QuantLib::LfmCovarianceParameterization, 612
- QuantLib::LfmCovarianceProxy, 613
- QuantLib::LfmHullWhiteParameterization, 614
- QuantLib::LfmSwaptionEngine, 615
- QuantLib::Libor, 616
- QuantLib::LiborForwardModel, 617
- QuantLib::LiborForwardModelProcess, 619
- QuantLib::LiborForwardModelProcess
  - apply, 620
  - covariance, 620
  - evolve, 620
- QuantLib::Linear, 621
- QuantLib::LinearInterpolation, 622
- QuantLib::LinearInterpolation
  - LinearInterpolation, 622
- QuantLib::LineSearch, 623
- QuantLib::Link, 625
- QuantLib::Link
  - Link, 625
  - linkTo, 626
- QuantLib::LmCorrelationModel, 627
- QuantLib::LmExponentialCorrelationModel, 628
- QuantLib::LmLinearExponentialVolatilityModel, 629
- QuantLib::LmVolatilityModel, 630
- QuantLib::LocalConstantVol, 631
- QuantLib::LocalVolCurve, 632
- QuantLib::LocalVolCurve
  - localVolImpl, 633
- QuantLib::LocalVolSurface, 634
- QuantLib::LocalVolTermStructure, 636
- QuantLib::LocalVolTermStructure
  - LocalVolTermStructure, 637
- QuantLib::LogLinear, 638
- QuantLib::LogLinearInterpolation, 639
- QuantLib::LogLinearInterpolation
  - LogLinearInterpolation, 639
- QuantLib::LTLCurrency, 640
- QuantLib::LUFCurrency, 641
- QuantLib::LVLCurrency, 642
- QuantLib::MakeMCDigitalEngine, 643
- QuantLib::MakeMCEuropeanEngine, 644
- QuantLib::MakeMCEuropeanHestonEngine, 645
- QuantLib::MakeSchedule, 646
- QuantLib::Matrix, 647
- QuantLib::Matrix
  - operator+=, 649
  - pseudoSqrt, 649
  - rankReducedSqrt, 649
- QuantLib::MCAmericanBasketEngine, 651
- QuantLib::MCBarrierEngine, 652
- QuantLib::MCBasketEngine, 654
- QuantLib::McCliquetOption, 656
- QuantLib::MCDigitalEngine, 657
- QuantLib::MCDiscreteArithmeticAPEngine, 658
- QuantLib::McDiscreteArithmeticASO, 660
- QuantLib::MCDiscreteAveragingAsianEngine, 661
- QuantLib::MCDiscreteGeometricAPEngine, 663
- QuantLib::McDiscreteArithmeticASO, 660
- QuantLib::MCEuropeanEngine, 664
- QuantLib::MCEuropeanHestonEngine, 665
- QuantLib::McEverest, 666
- QuantLib::McHimalaya, 667
- QuantLib::McMaxBasket, 668
- QuantLib::McPagoda, 669
- QuantLib::McPerformanceOption, 670

- QuantLib::McPricer, [671](#)
- QuantLib::McSimulation, [672](#)
- QuantLib::McSimulation
  - calculate, [673](#)
- QuantLib::MCVanillaEngine, [674](#)
- QuantLib::MersenneTwisterUniformRng, [675](#)
- QuantLib::MersenneTwisterUniformRng
  - MersenneTwisterUniformRng, [675](#)
  - next, [675](#)
- QuantLib::Merton76Process, [676](#)
- QuantLib::Merton76Process
  - apply, [677](#)
  - time, [677](#)
- QuantLib::Mexico, [678](#)
  - BMV, [679](#)
- QuantLib::Mexico
  - Market, [679](#)
- QuantLib::MixedScheme, [680](#)
- QuantLib::Money, [682](#)
  - AutomatedConversion, [683](#)
  - BaseCurrencyConversion, [683](#)
  - NoConversion, [683](#)
- QuantLib::Money
  - ConversionType, [683](#)
- QuantLib::MonotonicCubicSpline, [684](#)
- QuantLib::MonotonicCubicSpline
  - MonotonicCubicSpline, [684](#)
- QuantLib::MonteCarloModel, [685](#)
- QuantLib::MoreGreeks, [686](#)
- QuantLib::MoreInverseCumulativeNormal, [687](#)
- QuantLib::MTLCurrency, [688](#)
- QuantLib::MultiAssetOption, [689](#)
- QuantLib::MultiAssetOption
  - fetchResults, [690](#)
  - setupArguments, [690](#)
  - setupExpired, [690](#)
- QuantLib::MultiAssetOption::arguments, [691](#)
- QuantLib::MultiAssetOption::results, [692](#)
- QuantLib::MultiCubicSpline, [693](#)
- QuantLib::MultiPath, [694](#)
- QuantLib::MultiPathGenerator, [695](#)
- QuantLib::MultiVariate, [696](#)
- QuantLib::MXNCurrency, [697](#)
- QuantLib::NaturalCubicSpline, [698](#)
- QuantLib::NaturalCubicSpline
  - NaturalCubicSpline, [698](#)
- QuantLib::NaturalMonotonicCubicSpline, [699](#)
- QuantLib::NaturalMonotonicCubicSpline
  - NaturalMonotonicCubicSpline, [699](#)
- QuantLib::NeumannBC, [700](#)
- QuantLib::NeumannBC
  - applyAfterApplying, [700](#)
  - applyAfterSolving, [701](#)
  - applyBeforeApplying, [700](#)
  - applyBeforeSolving, [700](#)
  - setTime, [701](#)
- QuantLib::Newton, [702](#)
- QuantLib::NewtonSafe, [703](#)
- QuantLib::NewZealand, [704](#)
- QuantLib::NLGCurrency, [705](#)
- QuantLib::NoConstraint, [706](#)
- QuantLib::NOKCurrency, [707](#)
- QuantLib::NonLinearLeastSquare, [708](#)
- QuantLib::NormalDistribution, [709](#)
- QuantLib::Norway, [710](#)
- QuantLib::NPRCurrency, [711](#)
- QuantLib::Null, [712](#)
- QuantLib::NullCalendar, [713](#)
- QuantLib::NullCondition, [714](#)
- QuantLib::NullParameter, [715](#)
- QuantLib::NumericalMethod, [716](#)
- QuantLib::NumericalMethod
  - partialRollback, [716](#)
  - rollback, [716](#)
- QuantLib::NZDCurrency, [718](#)
- QuantLib::NZDLibor, [719](#)
- QuantLib::Observable, [720](#)
- QuantLib::Observable
  - notifyObservers, [720](#)
- QuantLib::ObservableValue, [721](#)
- QuantLib::Observer, [722](#)
- QuantLib::Observer
  - update, [723](#)
- QuantLib::OneAssetOption, [724](#)
- QuantLib::OneAssetOption
  - fetchResults, [725](#)
  - impliedVolatility, [725](#)
  - setupArguments, [725](#)
  - setupExpired, [726](#)
- QuantLib::OneAssetOption::arguments, [727](#)
- QuantLib::OneAssetOption::results, [728](#)
- QuantLib::OneAssetStrikedOption, [729](#)
- QuantLib::OneAssetStrikedOption
  - fetchResults, [729](#)
  - setupArguments, [729](#)
  - setupExpired, [730](#)
- QuantLib::OneDayCounter, [731](#)
- QuantLib::OneFactorAffineModel, [732](#)
- QuantLib::OneFactorModel, [733](#)
- QuantLib::OneFactorModel::ShortRateDynamics, [734](#)
- QuantLib::OneFactorModel::ShortRateTree, [735](#)
- QuantLib::OperatorFactory, [736](#)
- QuantLib::OptimizationMethod, [737](#)
- QuantLib::Option, [739](#)
- QuantLib::Option::arguments, [740](#)

- QuantLib::OrnsteinUhlenbeckProcess, 741
- QuantLib::OrnsteinUhlenbeckProcess
  - expectation, 741
  - stdDeviation, 741
  - variance, 742
- QuantLib::Parameter, 743
- QuantLib::ParameterImpl, 744
- QuantLib::ParCoupon, 745
- QuantLib::ParCoupon
  - amount, 746
  - update, 746
- QuantLib::Path, 747
- QuantLib::PathGenerator, 748
- QuantLib::PathPricer, 749
- QuantLib::Payoff, 750
- QuantLib::PercentageStrikePayoff, 751
- QuantLib::Period, 752
- QuantLib::PiecewiseConstantParameter, 753
- QuantLib::PiecewiseYieldCurve, 754
- QuantLib::PiecewiseYieldCurve
  - update, 755
- QuantLib::PKRCurrency, 756
- QuantLib::PlainVanillaPayoff, 757
- QuantLib::PLNCurrency, 758
- QuantLib::PoissonDistribution, 759
- QuantLib::Poland, 760
- QuantLib::PositiveConstraint, 761
- QuantLib::PriceCurve, 762
- QuantLib::PricingEngine, 763
- QuantLib::PrimeNumbers, 764
- QuantLib::Problem, 765
- QuantLib::PTECurrency, 767
- QuantLib::QuantoEngine, 768
- QuantLib::QuantoEngine
  - underlyingArgs, 769
- QuantLib::QuantoForwardVanillaOption, 770
- QuantLib::QuantoForwardVanillaOption
  - setupArguments, 771
- QuantLib::QuantoOptionArguments, 772
- QuantLib::QuantoOptionResults, 773
- QuantLib::QuantoTermStructure, 774
- QuantLib::QuantoVanillaOption, 776
- QuantLib::QuantoVanillaOption
  - fetchResults, 777
  - setupArguments, 777
  - setupExpired, 777
- QuantLib::Quote, 778
- QuantLib::RandomizedLDS, 779
- QuantLib::RandomizedLDS
  - nextRandomizer, 780
- QuantLib::RandomSequenceGenerator, 781
- QuantLib::RateHelper, 782
- QuantLib::RateHelper
  - latestDate, 783
  - setTermStructure, 783
  - update, 783
- QuantLib::Results, 784
- QuantLib::Ridder, 785
- QuantLib::ROLCurrency, 786
- QuantLib::Rounding, 787
  - Ceiling, 788
  - Closest, 788
  - Down, 788
  - Floor, 788
  - None, 788
  - Up, 788
- QuantLib::Rounding
  - Rounding, 788
  - Type, 787
- QuantLib::SalvagingAlgorithm, 789
- QuantLib::Sample, 790
- QuantLib::SampledCurve, 791
- QuantLib::SampledCurve
  - firstDerivativeAtCenter, 792
  - secondDerivativeAtCenter, 792
  - valueAtCenter, 792
- QuantLib::SARCurrency, 793
- QuantLib::SaudiArabia, 794
- QuantLib::Schedule, 795
- QuantLib::Secant, 796
- QuantLib::SeedGenerator, 797
- QuantLib::SegmentIntegral, 798
- QuantLib::SEKCurrency, 799
- QuantLib::SequenceStatistics, 800
- QuantLib::Settings, 802
- QuantLib::Settings
  - evaluationDate, 802
- QuantLib::SGDCurrency, 804
- QuantLib::Short, 805
- QuantLib::Short
  - amount, 805
- QuantLib::Short< ParCoupon >, 806
- QuantLib::ShortRateModel, 807
- QuantLib::ShortRateModel
  - calibrate, 808
  - update, 808
- QuantLib::ShoutCondition, 809
- QuantLib::SimpleCashFlow, 810
- QuantLib::SimpleCashFlow
  - amount, 810
  - date, 810
- QuantLib::SimpleDayCounter, 812
- QuantLib::SimpleQuote, 813
- QuantLib::Simplex, 814
- QuantLib::Simplex
  - Simplex, 814
- QuantLib::SimpsonIntegral, 815
- QuantLib::Singapore, 816



- SGX, [817](#)
- QuantLib::Singapore
  - Market, [817](#)
- QuantLib::SingleAssetOption, [818](#)
- QuantLib::SingleAssetOption
  - impliedVolatility, [819](#)
- QuantLib::Singleton, [820](#)
- QuantLib::SingleVariate, [821](#)
- QuantLib::SITCurrency, [822](#)
- QuantLib::SKKCurrency, [823](#)
- QuantLib::Slovakia, [824](#)
  - BSSE, [825](#)
- QuantLib::Slovakia
  - Market, [825](#)
- QuantLib::SobolRsg, [826](#)
- QuantLib::SobolRsg
  - SobolRsg, [827](#)
- QuantLib::Solver1D, [828](#)
- QuantLib::Solver1D
  - setMaxEvaluations, [829](#)
  - solve, [829](#)
- QuantLib::SouthAfrica, [830](#)
- QuantLib::SouthKorea, [831](#)
  - KRX, [832](#)
- QuantLib::SouthKorea
  - Market, [832](#)
- QuantLib::SquareRootProcess, [833](#)
- QuantLib::StatsHolder, [834](#)
- QuantLib::SteepestDescent, [835](#)
- QuantLib::step\_iterator, [836](#)
- QuantLib::StepCondition, [837](#)
- QuantLib::StepConditionSet, [838](#)
- QuantLib::StochasticProcess, [839](#)
- QuantLib::StochasticProcess
  - apply, [841](#)
  - covariance, [840](#)
  - evolve, [840](#)
  - expectation, [840](#)
  - stdDeviation, [840](#)
  - time, [841](#)
  - update, [841](#)
- QuantLib::StochasticProcess1D, [842](#)
- QuantLib::StochasticProcess1D
  - apply, [843](#)
  - evolve, [843](#)
  - expectation, [843](#)
  - stdDeviation, [843](#)
  - variance, [843](#)
- QuantLib::StochasticProcess1D::discretization, [844](#)
- QuantLib::StochasticProcess::discretization, [845](#)
- QuantLib::StochasticProcessArray, [846](#)
- QuantLib::StochasticProcessArray
  - apply, [847](#)
  - covariance, [847](#)
  - expectation, [847](#)
  - stdDeviation, [847](#)
  - time, [847](#)
- QuantLib::Stock, [848](#)
- QuantLib::Stock
  - performCalculations, [848](#)
- QuantLib::StrikedTypePayoff, [849](#)
- QuantLib::StulzEngine, [850](#)
- QuantLib::SuperSharePayoff, [851](#)
- QuantLib::SVD, [852](#)
- QuantLib::Swap, [853](#)
- QuantLib::Swap
  - performCalculations, [854](#)
  - sensitivity, [854](#)
  - setupExpired, [854](#)
- QuantLib::SwapRateHelper, [855](#)
- QuantLib::SwapRateHelper
  - latestDate, [856](#)
  - setTermStructure, [856](#)
  - SwapRateHelper, [856](#)
- QuantLib::Swaption, [858](#)
- QuantLib::Swaption
  - setupArguments, [859](#)
- QuantLib::Swaption::arguments, [860](#)
- QuantLib::Swaption::results, [861](#)
- QuantLib::SwaptionHelper, [862](#)
- QuantLib::SwaptionHelper
  - SwaptionHelper, [862](#)
- QuantLib::SwaptionVolatilityMatrix, [863](#)
- QuantLib::SwaptionVolatilityStructure, [865](#)
- QuantLib::SwaptionVolatilityStructure
  - SwaptionVolatilityStructure, [866](#)
- QuantLib::Sweden, [867](#)
- QuantLib::Switzerland, [868](#)
- QuantLib::SymmetricSchurDecomposition, [869](#)
- QuantLib::SymmetricSchurDecomposition
  - SymmetricSchurDecomposition, [869](#)
- QuantLib::TabulatedGaussLegendre, [870](#)
- QuantLib::Taiwan, [871](#)
  - TSEC, [872](#)
- QuantLib::Taiwan
  - Market, [872](#)
- QuantLib::TARGET, [873](#)
- QuantLib::TermStructure, [874](#)
- QuantLib::TermStructure
  - TermStructure, [875](#)
  - update, [875](#)
- QuantLib::TermStructureConsistentModel, [876](#)
- QuantLib::TermStructureFittingParameter, [877](#)
- QuantLib::THBCurrency, [878](#)
- QuantLib::Thirty360, [879](#)

- QuantLib::Tian, 880
- QuantLib::Tibor, 881
- QuantLib::TimeBasket, 882
- QuantLib::TimeGrid, 883
- QuantLib::TimeGrid
  - findIndex, 884
  - TimeGrid, 884
- QuantLib::TqrEigenDecomposition, 885
- QuantLib::TransformedGrid, 886
- QuantLib::TrapezoidIntegral, 887
- QuantLib::Tree, 889
- QuantLib::TreeCapFloorEngine, 890
- QuantLib::TreeSwaptionEngine, 891
- QuantLib::TreeVanillaSwapEngine, 892
- QuantLib::TridiagonalOperator, 893
- QuantLib::TridiagonalOperator::TimeSetter, 895
- QuantLib::Trigeorgis, 896
- QuantLib::TrinomialTree, 897
- QuantLib::TRLCurrency, 898
- QuantLib::TRLibor, 899
- QuantLib::TRYCurrency, 900
- QuantLib::TsiveriotisFernandesLattice, 901
- QuantLib::TsiveriotisFernandesLattice
  - partialRollback, 901
  - rollback, 901
- QuantLib::TTDCurrency, 903
- QuantLib::Turkey, 904
- QuantLib::TWDCurrency, 905
- QuantLib::TwoFactorModel, 906
- QuantLib::TwoFactorModel::ShortRateDynamics, 907
- QuantLib::TwoFactorModel::ShortRateTree, 908
- QuantLib::TypePayoff, 909
- QuantLib::Ukraine, 910
  - USE, 911
- QuantLib::Ukraine
  - Market, 911
- QuantLib::UnitedKingdom, 912
  - Exchange, 913
  - Settlement, 913
- QuantLib::UnitedKingdom
  - Market, 913
- QuantLib::UnitedStates, 914
  - Exchange, 916
  - NYSE, 916
  - Settlement, 916
- QuantLib::UnitedStates
  - Market, 916
- QuantLib::UpFrontIndexedCoupon, 917
- QuantLib::UpRounding, 919
- QuantLib::USDCurrency, 920
- QuantLib::USDLibor, 921
- QuantLib::Value, 922
- QuantLib::VanillaOption, 923
- QuantLib::VanillaOption::engine, 924
- QuantLib::VanillaSwap, 925
- QuantLib::VanillaSwap
  - fetchResults, 926
  - setupArguments, 926
  - VanillaSwap, 926
- QuantLib::VanillaSwap::arguments, 927
- QuantLib::VanillaSwap::results, 928
- QuantLib::Vasicek, 929
- QuantLib::Vasicek::Dynamics, 931
- QuantLib::VEBCurrency, 932
- QuantLib::Visitor, 933
- QuantLib::Xibor, 934
- QuantLib::Xibor
  - fixing, 935
  - name, 935
  - update, 935
- QuantLib::YieldTermStructure, 936
- QuantLib::YieldTermStructure
  - discount, 938
  - forwardRate, 938
  - parRate, 938
  - YieldTermStructure, 938
  - zeroRate, 938
- QuantLib::ZARCurrency, 940
- QuantLib::ZeroCondition, 941
- QuantLib::ZeroCouponBond, 942
- QuantLib::ZeroSpreadedTermStructure, 943
- QuantLib::ZeroYield, 945
- QuantLib::ZeroYieldStructure, 946
- QuantLib::ZeroYieldStructure
  - discountImpl, 946
- QuantLib::Zibor, 948
- Quanto option engines, 107
- Quarterly
  - datetime, 95
- rankReducedSqrt
  - QuantLib::Matrix, 649
- recalculate
  - QuantLib::LazyObject, 604
- regret
  - QuantLib::GenericRiskStatistics, 503
- removeHoliday
  - QuantLib::Calendar, 263
- reset
  - QuantLib::DiscretizedAsset, 372
  - QuantLib::DiscretizedDiscountBond, 374
  - QuantLib::DiscretizedOption, 375
- rollback
  - QuantLib::FiniteDifferenceModel, 436
  - QuantLib::Lattice, 599

- QuantLib::NumericalMethod, 716
- QuantLib::TsiveriotisFernandesLattice, 901
- Rounding
  - QuantLib::Rounding, 788
- SecondDerivative
  - QuantLib::CubicSpline, 337
- secondDerivativeAtCenter
  - QuantLib::SampledCurve, 792
- Semiannual
  - datetime, 95
- semiDeviation
  - QuantLib::GenericRiskStatistics, 502
- semiVariance
  - QuantLib::GenericRiskStatistics, 502
- sensitivity
  - QuantLib::Swap, 854
- sequencestatistics.hpp
  - DEFINE\_SEQUENCE\_STAT\_CONST\_-METHOD\_DOUBLE, 1164
  - DEFINE\_SEQUENCE\_STAT\_CONST\_-METHOD\_VOID, 1164
- setMaxEvaluations
  - QuantLib::Solver1D, 829
- setPricingEngine
  - QuantLib::Instrument, 556
- setTermStructure
  - QuantLib::DepositRateHelper, 359
  - QuantLib::FixedCouponBondHelper, 441
  - QuantLib::FraRateHelper, 468
  - QuantLib::RateHelper, 783
  - QuantLib::SwapRateHelper, 856
- setTime
  - QuantLib::BoundaryCondition, 247
  - QuantLib::DirichletBC, 363
  - QuantLib::NeumannBC, 701
- Settlement
  - QuantLib::Germany, 508
  - QuantLib::Italy, 582
  - QuantLib::UnitedKingdom, 913
  - QuantLib::UnitedStates, 916
- setupArguments
  - QuantLib::BarrierOption, 192
  - QuantLib::BasketOption, 196
  - QuantLib::CapFloor, 273
  - QuantLib::CliquetOption, 298
  - QuantLib::ContinuousAveragingAsian-Option, 315
  - QuantLib::DiscreteAveragingAsian-Option, 367
  - QuantLib::DividendVanillaOption, 381
  - QuantLib::ForwardVanillaOption, 465
  - QuantLib::Instrument, 556
  - QuantLib::MultiAssetOption, 690
  - QuantLib::OneAssetOption, 725
  - QuantLib::OneAssetStrikedOption, 729
  - QuantLib::QuantoForwardVanillaOption, 771
  - QuantLib::QuantoVanillaOption, 777
  - QuantLib::Swaption, 859
  - QuantLib::VanillaSwap, 926
- setupExpired
  - QuantLib::Instrument, 557
  - QuantLib::MultiAssetOption, 690
  - QuantLib::OneAssetOption, 726
  - QuantLib::OneAssetStrikedOption, 730
  - QuantLib::QuantoVanillaOption, 777
  - QuantLib::Swap, 854
- SGX
  - QuantLib::Singapore, 817
- Short-rate modelling framework, 119
- shortfall
  - QuantLib::GenericRiskStatistics, 503
- Side
  - QuantLib::BoundaryCondition, 246
- Simplex
  - QuantLib::Simplex, 814
- skewness
  - QuantLib::GeneralStatistics, 498
  - QuantLib::IncrementalStatistics, 545
- SobolRsg
  - QuantLib::SobolRsg, 827
- solve
  - QuantLib::Solver1D, 829
- standardDeviation
  - QuantLib::GeneralStatistics, 498
  - QuantLib::IncrementalStatistics, 544
- standardDeviations
  - QuantLib::CovarianceDecomposition, 328
- stdDeviation
  - QuantLib::OrnsteinUhlenbeckProcess, 741
  - QuantLib::StochasticProcess, 840
  - QuantLib::StochasticProcess1D, 843
  - QuantLib::StochasticProcessArray, 847
- SwapRateHelper
  - QuantLib::SwapRateHelper, 856
- Swaption engines, 108
- SwaptionHelper
  - QuantLib::SwaptionHelper, 862
- SwaptionVolatilityStructure
  - QuantLib::SwaptionVolatilityStructure, 866
- SymmetricSchurDecomposition
  - QuantLib::SymmetricSchur-Decomposition, 869
- targetValueAndGradient



- QuantLib::LeastSquareProblem, 606
- Template capabilities, 144
- templateMacros
  - QL\_TYPENAME, 144
- Term structures, 137
- TermStructure
  - QuantLib::TermStructure, 875
- time
  - QuantLib::BlackScholesProcess, 229
  - QuantLib::HestonProcess, 516
  - QuantLib::Merton76Process, 677
  - QuantLib::StochasticProcess, 841
  - QuantLib::StochasticProcessArray, 847
- TimeGrid
  - QuantLib::TimeGrid, 884
- topPercentile
  - QuantLib::GeneralStatistics, 499
- TSEC
  - QuantLib::Taiwan, 872
- Type
  - QuantLib::ExchangeRate, 413
  - QuantLib::Rounding, 787
- Unadjusted
  - datetime, 94
- underlyingArgs
  - QuantLib::QuantoEngine, 769
- unfreeze
  - QuantLib::LazyObject, 604
- Up
  - QuantLib::Rounding, 788
- update
  - QuantLib::AffineTermStructure, 158
  - QuantLib::BlackModel, 225
  - QuantLib::BlackScholesProcess, 229
  - QuantLib::CalibrationHelper, 269
  - QuantLib::CapVolatilityVector, 284
  - QuantLib::CompositeQuote, 307
  - QuantLib::DerivedQuote, 361
  - QuantLib::ExtendedDiscountCurve, 424
  - QuantLib::FlatForward, 447
  - QuantLib::GenericModelEngine, 501
  - QuantLib::IndexedCoupon, 548
  - QuantLib::LatticeShortRateModelEngine, 602
  - QuantLib::LazyObject, 603
  - QuantLib::Observer, 723
  - QuantLib::ParCoupon, 746
  - QuantLib::PiecewiseYieldCurve, 755
  - QuantLib::RateHelper, 783
  - QuantLib::ShortRateModel, 808
  - QuantLib::StochasticProcess, 841
  - QuantLib::TermStructure, 875
  - QuantLib::Xibor, 935
- USE
  - QuantLib::Ukraine, 911
- Utilities, 139
- valueAtCenter
  - QuantLib::SampledCurve, 792
- valueAtRisk
  - QuantLib::GenericRiskStatistics, 503
- Vanilla option engines, 109
- vanillaengines
  - FDAmericanEngine, 110
  - FDDividendAmericanEngine, 110
  - FDDividendEuropeanEngine, 111
  - FDDividendShoutEngine, 111
  - FDShoutEngine, 111
- VanillaSwap
  - QuantLib::VanillaSwap, 926
- variance
  - QuantLib::EulerDiscretization, 404
  - QuantLib::GeneralStatistics, 498
  - QuantLib::IncrementalStatistics, 544
  - QuantLib::OrnsteinUhlenbeckProcess, 742
  - QuantLib::StochasticProcess1D, 843
- variances
  - QuantLib::CovarianceDecomposition, 328
- Weekday
  - datetime, 95
- Xetra
  - QuantLib::Germany, 508
- yield
  - QuantLib::Bond, 244, 245
- YieldTermStructure
  - QuantLib::YieldTermStructure, 938
- yieldtermstructures
  - DiscountCurve, 138
- zeroRate
  - QuantLib::YieldTermStructure, 938
- zeroYieldImpl
  - QuantLib::CompoundForward, 309
  - QuantLib::ExtendedDiscountCurve, 424
  - QuantLib::ForwardRateStructure, 461
  - QuantLib::InterpolatedForwardCurve, 565