

QuantLib

An open source library for quantitative finance

Version 0.3.11

Generated by Doxygen 1.4.5

17 Oct 2005

Contents

1	Getting started	1
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	20
1.9	Additional resources	39
1.10	The QuantLib Group	40
1.11	QuantLib License	41
2	QuantLib Module Index	43
2.1	QuantLib Modules	43
3	QuantLib Hierarchical Index	45
3.1	QuantLib Class Hierarchy	45
4	QuantLib Class Index	61
4.1	QuantLib Class List	61
5	QuantLib File Index	73
5.1	QuantLib File List	73
6	QuantLib Module Documentation	83
6.1	Numeric types	83
6.2	Currencies and FX rates	85
6.3	Date and time calculations	89
6.4	Calendars	92

6.5	Day counters	94
6.6	Pricing engines	95
6.7	Asian option engines	96
6.8	Barrier option engines	97
6.9	Basket option engines	98
6.10	Cap/floor engines	99
6.11	Cliquet option engines	100
6.12	Forward option engines	101
6.13	Quanto option engines	102
6.14	Swaption engines	103
6.15	Vanilla option engines	104
6.16	Finite-differences framework	106
6.17	Short-rate modelling framework	112
6.18	Financial instruments	115
6.19	Lattice methods	118
6.20	Math tools	121
6.21	Monte Carlo framework	123
6.22	Design patterns	129
6.23	Term structures	130
6.24	Utilities	132
6.25	QuantLib macros	134
6.26	Generic macros	135
6.27	Numeric limits	136
6.28	Template capabilities	137
6.29	Iterator support	138
6.30	Output manipulators	139
6.31	Debugging macros	140
7	QuantLib Class Documentation	143
7.1	Actual360 Class Reference	143
7.2	Actual365Fixed Class Reference	144
7.3	ActualActual Class Reference	145
7.4	AcyclicVisitor Class Reference	146
7.5	AdditiveEQPBinoomialTree Class Reference	147
7.6	AffineModel Class Reference	148
7.7	AffineTermStructure Class Reference	149
7.8	AmericanCondition Class Reference	151

7.9	AmericanExercise Class Reference	152
7.10	AmericanPayoffAtExpiry Class Reference	153
7.11	AmericanPayoffAtHit Class Reference	154
7.12	AnalyticBarrierEngine Class Reference	155
7.13	AnalyticCapFloorEngine Class Reference	156
7.14	AnalyticCliquetEngine Class Reference	157
7.15	AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference	158
7.16	AnalyticDigitalAmericanEngine Class Reference	159
7.17	AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference	160
7.18	AnalyticDividendEuropeanEngine Class Reference	161
7.19	AnalyticEuropeanEngine Class Reference	162
7.20	AnalyticHestonEngine Class Reference	163
7.21	AnalyticPerformanceEngine Class Reference	164
7.22	Arguments Class Reference	165
7.23	ArmijoLineSearch Class Reference	166
7.24	Array Class Reference	167
7.25	ARSCurrency Class Reference	170
7.26	AssetOrNothingPayoff Class Reference	171
7.27	ATSCurrency Class Reference	172
7.28	AUDCurrency Class Reference	173
7.29	AUDLibor Class Reference	174
7.30	Average Struct Reference	175
7.31	BackwardFlat Class Reference	176
7.32	BackwardFlatInterpolation Class Reference	177
7.33	BaroneAdesiWhaleyApproximationEngine Class Reference	178
7.34	Barrier Struct Reference	179
7.35	BarrierOption Class Reference	180
7.36	BarrierOption::arguments Class Reference	182
7.37	BarrierOption::engine Class Reference	183
7.38	BasketOption Class Reference	184
7.39	BasketOption::arguments Class Reference	186
7.40	BasketOption::engine Class Reference	187
7.41	BatesEngine Class Reference	188
7.42	BatesModel Class Reference	190
7.43	BDTCurrency Class Reference	191
7.44	BEFCurrency Class Reference	192

7.45	Beijing Class Reference	193
7.46	BermudanExercise Class Reference	194
7.47	BGLCurrency Class Reference	195
7.48	Bicubic Class Reference	196
7.49	BicubicSpline Class Reference	197
7.50	Bilinear Class Reference	198
7.51	BilinearInterpolation Class Reference	199
7.52	BinomialDistribution Class Reference	200
7.53	BinomialTree Class Template Reference	201
7.54	BinomialVanillaEngine Class Template Reference	202
7.55	Bisection Class Reference	203
7.56	BivariateCumulativeNormalDistributionDr78 Class Reference	204
7.57	BivariateCumulativeNormalDistributionWe04DP Class Reference	205
7.58	BjerkstrandStenslandApproximationEngine Class Reference	206
7.59	BlackCapFloorEngine Class Reference	207
7.60	BlackConstantVol Class Reference	208
7.61	BlackFormula Class Reference	210
7.62	BlackKarasinski Class Reference	212
7.63	BlackKarasinski::Dynamics Class Reference	213
7.64	BlackModel Class Reference	214
7.65	BlackScholesLattice Class Template Reference	216
7.66	BlackScholesProcess Class Reference	217
7.67	BlackSwaptionEngine Class Reference	219
7.68	BlackVarianceCurve Class Reference	220
7.69	BlackVarianceSurface Class Reference	222
7.70	BlackVarianceTermStructure Class Reference	224
7.71	BlackVolatilityTermStructure Class Reference	226
7.72	BlackVolTermStructure Class Reference	228
7.73	Bombay Class Reference	231
7.74	Bond Class Reference	232
7.75	BoundaryCondition Class Template Reference	235
7.76	BoundaryConstraint Class Reference	237
7.77	BoxMullerGaussianRng Class Template Reference	238
7.78	BPSBasketCalculator Class Reference	239
7.79	BPSCalculator Class Reference	240
7.80	Bratislava Class Reference	241

7.81	Brent Class Reference	242
7.82	Bridge Class Template Reference	243
7.83	BRLCurrency Class Reference	244
7.84	BrownianBridge Class Template Reference	245
7.85	BSMOperator Class Reference	246
7.86	BSMTermOperator Class Reference	247
7.87	Budapest Class Reference	248
7.88	BYRCurrency Class Reference	249
7.89	CADCurrency Class Reference	250
7.90	CADLibor Class Reference	251
7.91	Calendar Class Reference	252
7.92	Calendar::WesternImpl Class Reference	257
7.93	CalendarImpl Class Reference	258
7.94	CalibrationHelper Class Reference	259
7.95	Cap Class Reference	261
7.96	CapFloor Class Reference	262
7.97	CapFloor::arguments Class Reference	264
7.98	CapFloor::results Class Reference	265
7.99	CapletConstantVolatility Class Reference	266
7.100	CapletLiborMarketModelProcess Class Reference	268
7.101	CapletVolatilityStructure Class Reference	270
7.102	CapVolatilityStructure Class Reference	272
7.103	CapVolatilityVector Class Reference	274
7.104	CashFlow Class Reference	276
7.105	Cashflows Class Reference	277
7.106	CashOrNothingPayoff Class Reference	279
7.107	Cdor Class Reference	280
7.108	CeilingTruncation Class Reference	281
7.109	CHFCurrency Class Reference	282
7.110	CHFLibor Class Reference	283
7.111	CLGaussianRng Class Template Reference	284
7.112	CliquetOption Class Reference	285
7.113	CliquetOption::arguments Class Reference	287
7.114	CliquetOption::engine Class Reference	288
7.115	ClosestRounding Class Reference	289
7.116	CLPCurrency Class Reference	290

7.117	CNYCurrency Class Reference	291
7.118	Collar Class Reference	292
7.119	Composite Class Template Reference	293
7.120	CompositeConstraint Class Reference	294
7.121	CompositeQuote Class Template Reference	295
7.122	CompoundForward Class Reference	296
7.123	ConjugateGradient Class Reference	298
7.124	ConstantParameter Class Reference	299
7.125	Constraint Class Reference	300
7.126	ConstraintImpl Class Reference	301
7.127	ContinuousAveragingAsianOption Class Reference	302
7.128	ContinuousAveragingAsianOption::arguments Class Reference	304
7.129	ContinuousAveragingAsianOption::engine Class Reference	305
7.130	ConvergenceStatistics Class Template Reference	306
7.131	ConvertibleBond::option Class Reference	307
7.132	ConvertibleBond::option::arguments Class Reference	309
7.133	ConvertibleBond::option::engine Class Reference	310
7.134	COPCurrency Class Reference	311
7.135	Copenhagen Class Reference	312
7.136	CostFunction Class Reference	313
7.137	Coupon Class Reference	314
7.138	CovarianceDecomposition Class Reference	316
7.139	CoxIngersollRoss Class Reference	317
7.140	CoxIngersollRoss::Dynamics Class Reference	319
7.141	CoxRossRubinstein Class Reference	320
7.142	CrankNicolson Class Template Reference	321
7.143	Cubic Class Reference	323
7.144	CubicSpline Class Reference	324
7.145	CumulativeBinomialDistribution Class Reference	326
7.146	CumulativeNormalDistribution Class Reference	327
7.147	CumulativePoissonDistribution Class Reference	328
7.148	CuriouslyRecurringTemplate Class Template Reference	329
7.149	Currency Class Reference	330
7.150	CYPCurrency Class Reference	334
7.151	CZKCurrency Class Reference	335
7.152	Date Class Reference	336

7.153	DayCounter Class Reference	340
7.154	DayCounterImpl Class Reference	342
7.155	DEMCurrency Class Reference	343
7.156	DepositRateHelper Class Reference	344
7.157	DerivedQuote Class Template Reference	346
7.158	DirichletBC Class Reference	347
7.159	Discount Struct Reference	349
7.160	DiscrepancyStatistics Class Reference	350
7.161	DiscreteAveragingAsianOption Class Reference	351
7.162	DiscreteAveragingAsianOption::arguments Class Reference	353
7.163	DiscreteAveragingAsianOption::engine Class Reference	354
7.164	DiscreteGeometricASO Class Reference	355
7.165	DiscretizedAsset Class Reference	356
7.166	DiscretizedDiscountBond Class Reference	359
7.167	DiscretizedOption Class Reference	360
7.168	Disposable Class Template Reference	362
7.169	DividendVanillaOption Class Reference	363
7.170	DividendVanillaOption::arguments Class Reference	365
7.171	DividendVanillaOption::engine Class Reference	366
7.172	DKKCurrency Class Reference	367
7.173	DKKLibor Class Reference	368
7.174	DMinus Class Reference	369
7.175	DownRounding Class Reference	370
7.176	DPlus Class Reference	371
7.177	DPlusDMinus Class Reference	372
7.178	DriftTermStructure Class Reference	373
7.179	Duration Struct Reference	375
7.180	DZero Class Reference	376
7.181	EarlyExercise Class Reference	377
7.182	EEKCurrency Class Reference	378
7.183	EndCriteria Class Reference	379
7.184	EqualJumpsBinomialTree Class Template Reference	381
7.185	EqualProbabilitiesBinomialTree Class Template Reference	382
7.186	Error Class Reference	383
7.187	ErrorFunction Class Reference	384
7.188	ESPCurrency Class Reference	385

7.189	EulerDiscretization Class Reference	386
7.190	EURCurrency Class Reference	388
7.191	Euribor Class Reference	389
7.192	EURLibor Class Reference	390
7.193	EuropeanExercise Class Reference	391
7.194	EuropeanOption Class Reference	392
7.195	ExchangeRate Class Reference	393
7.196	ExchangeRateManager Class Reference	395
7.197	Exercise Class Reference	397
7.198	ExplicitEuler Class Template Reference	398
7.199	ExtendedCoxIngersollRoss Class Reference	400
7.200	ExtendedCoxIngersollRoss::Dynamics Class Reference	402
7.201	ExtendedCoxIngersollRoss::FittingParameter Class Reference	403
7.202	ExtendedDiscountCurve Class Reference	404
7.203	Extrapolator Class Reference	406
7.204	Factorial Class Reference	407
7.205	FalsePosition Class Reference	408
7.206	FaureRsg Class Reference	409
7.207	FDAmericanEngine Class Reference	410
7.208	FDBermudanEngine Class Reference	411
7.209	FDDividendAmericanEngine Class Reference	412
7.210	FDDividendEngine Class Reference	413
7.211	FDDividendEuropeanEngine Class Reference	414
7.212	FDDividendShoutEngine Class Reference	415
7.213	FDEuropeanEngine Class Reference	416
7.214	FDShoutEngine Class Reference	417
7.215	FDStepConditionEngine Class Reference	418
7.216	FDVanillaEngine Class Reference	419
7.217	FIMCurrency Class Reference	421
7.218	FiniteDifferenceModel Class Template Reference	422
7.219	FixedCouponBond Class Reference	423
7.220	FixedCouponBondHelper Class Reference	424
7.221	FixedRateCoupon Class Reference	426
7.222	FlatForward Class Reference	428
7.223	FloatingRateBond Class Reference	430
7.224	FloatingRateCoupon Class Reference	431

7.225	Floor Class Reference	433
7.226	FloorTruncation Class Reference	434
7.227	ForwardEngine Class Template Reference	435
7.228	ForwardFlat Class Reference	436
7.229	ForwardFlatInterpolation Class Reference	437
7.230	ForwardOptionArguments Class Template Reference	438
7.231	ForwardPerformanceEngine Class Template Reference	439
7.232	ForwardRate Struct Reference	440
7.233	ForwardRateStructure Class Reference	441
7.234	ForwardSpreadedTermStructure Class Reference	443
7.235	ForwardVanillaOption Class Reference	445
7.236	FraRateHelper Class Reference	447
7.237	FRFCurrency Class Reference	449
7.238	FuturesRateHelper Class Reference	450
7.239	G2 Class Reference	451
7.240	G2::FittingParameter Class Reference	453
7.241	G2SwaptionEngine Class Reference	454
7.242	GammaFunction Class Reference	455
7.243	GapPayoff Class Reference	456
7.244	GaussChebyshev2thIntegration Class Reference	457
7.245	GaussChebyshevIntegration Class Reference	458
7.246	GaussGegenbauerIntegration Class Reference	459
7.247	GaussHermiteIntegration Class Reference	460
7.248	GaussHermitePolynomial Class Reference	461
7.249	GaussHyperbolicIntegration Class Reference	462
7.250	GaussHyperbolicPolynomial Class Reference	463
7.251	GaussianOrthogonalPolynomial Class Reference	464
7.252	GaussianQuadrature Class Reference	465
7.253	GaussianStatistics Class Template Reference	466
7.254	GaussJacobiIntegration Class Reference	469
7.255	GaussJacobiPolynomial Class Reference	470
7.256	GaussLaguerreIntegration Class Reference	471
7.257	GaussLaguerrePolynomial Class Reference	472
7.258	GaussLegendreIntegration Class Reference	473
7.259	GBPCurrency Class Reference	474
7.260	GBPLibor Class Reference	475

7.261	GeneralStatistics Class Reference	476
7.262	GenericEngine Class Template Reference	479
7.263	GenericModelEngine Class Template Reference	480
7.264	GenericRiskStatistics Class Template Reference	481
7.265	GeometricBrownianMotionProcess Class Reference	484
7.266	Germany Class Reference	485
7.267	GRDCurrency Class Reference	488
7.268	Greeks Class Reference	489
7.269	HaltonRsg Class Reference	490
7.270	Handle Class Template Reference	491
7.271	Helsinki Class Reference	492
7.272	HestonModel Class Reference	493
7.273	HestonModelHelper Class Reference	494
7.274	HestonProcess Class Reference	495
7.275	History Class Reference	497
7.276	History::const_iterator Class Reference	500
7.277	History::Entry Class Reference	501
7.278	HKDCurrency Class Reference	502
7.279	HongKong Class Reference	503
7.280	HUFCurrency Class Reference	504
7.281	HullWhite Class Reference	505
7.282	HullWhite::Dynamics Class Reference	507
7.283	HullWhite::FittingParameter Class Reference	508
7.284	IEPCurrency Class Reference	509
7.285	ILSCurrency Class Reference	510
7.286	IMM Struct Reference	511
7.287	ImplicitEuler Class Template Reference	512
7.288	ImpliedTermStructure Class Reference	513
7.289	ImpliedVolTermStructure Class Reference	515
7.290	InArrearIndexedCoupon Class Reference	517
7.291	IncrementalStatistics Class Reference	519
7.292	Index Class Reference	522
7.293	IndexedCoupon Class Reference	523
7.294	IndexManager Class Reference	525
7.295	INRCurrency Class Reference	526
7.296	Instrument Class Reference	527

7.297	IntegralEngine Class Reference	530
7.298	InterestRate Class Reference	531
7.299	InterpolatedDiscountCurve Class Template Reference	534
7.300	InterpolatedForwardCurve Class Template Reference	536
7.301	InterpolatedZeroCurve Class Template Reference	538
7.302	Interpolation Class Reference	540
7.303	Interpolation2D Class Reference	541
7.304	Interpolation2D::templateImpl Class Template Reference	542
7.305	Interpolation2DImpl Class Reference	543
7.306	Interpolation::templateImpl Class Template Reference	544
7.307	InterpolationImpl Class Reference	545
7.308	InverseCumulativeNormal Class Reference	546
7.309	InverseCumulativePoisson Class Reference	547
7.310	InverseCumulativeRng Class Template Reference	548
7.311	InverseCumulativeRsg Class Template Reference	549
7.312	IQDCurrency Class Reference	550
7.313	IRRCurrency Class Reference	551
7.314	ISKCurrency Class Reference	552
7.315	Istanbul Class Reference	553
7.316	Italy Class Reference	554
7.317	ITLCurrency Class Reference	556
7.318	JamshidianSwaptionEngine Class Reference	557
7.319	JarrowRudd Class Reference	558
7.320	Jibar Class Reference	559
7.321	Johannesburg Class Reference	560
7.322	JointCalendar Class Reference	561
7.323	JPYCurrency Class Reference	562
7.324	JPYLibor Class Reference	563
7.325	JumpDiffusionEngine Class Reference	564
7.326	JuQuadraticApproximationEngine Class Reference	565
7.327	KnuthUniformRng Class Reference	566
7.328	KronrodIntegral Class Reference	567
7.329	KRWCurrency Class Reference	568
7.330	KWDCurrency Class Reference	569
7.331	Lattice Class Template Reference	570
7.332	Lattice1D Class Template Reference	572

7.333	Lattice2D Class Template Reference	573
7.334	LatticeShortRateModelEngine Class Template Reference	574
7.335	LazyObject Class Reference	575
7.336	LeastSquareFunction Class Reference	577
7.337	LeastSquareProblem Class Reference	578
7.338	LecuyerUniformRng Class Reference	579
7.339	LeisenReimer Class Reference	580
7.340	LexicographicalView Class Template Reference	581
7.341	Libor Class Reference	583
7.342	Linear Class Reference	584
7.343	LinearInterpolation Class Reference	585
7.344	LineSearch Class Reference	586
7.345	Link Class Template Reference	588
7.346	LocalConstantVol Class Reference	590
7.347	LocalVolCurve Class Reference	591
7.348	LocalVolSurface Class Reference	593
7.349	LocalVolTermStructure Class Reference	595
7.350	LogLinear Class Reference	597
7.351	LogLinearInterpolation Class Reference	598
7.352	LTLCurrency Class Reference	599
7.353	LUFCurrency Class Reference	600
7.354	LVLCurrency Class Reference	601
7.355	MakeMCDigitalEngine Class Template Reference	602
7.356	MakeMCEuropeanEngine Class Template Reference	603
7.357	MakeMCEuropeanHestonEngine Class Template Reference	604
7.358	MakeSchedule Class Reference	605
7.359	Matrix Class Reference	606
7.360	MCAmericanBasketEngine Class Reference	610
7.361	MCBarrierEngine Class Template Reference	611
7.362	MCBasketEngine Class Template Reference	613
7.363	McCliquetOption Class Reference	615
7.364	MCDigitalEngine Class Template Reference	616
7.365	MCDiscreteArithmeticAPEngine Class Template Reference	617
7.366	McDiscreteArithmeticASO Class Reference	619
7.367	MCDiscreteAveragingAsianEngine Class Template Reference	620
7.368	MCDiscreteGeometricAPEngine Class Template Reference	622

7.369	MCEuropeanEngine Class Template Reference	623
7.370	MCEuropeanHestonEngine Class Template Reference	624
7.371	McEverest Class Reference	625
7.372	MCHestonEngine Class Template Reference	626
7.373	McHimalaya Class Reference	627
7.374	McMaxBasket Class Reference	628
7.375	McPagoda Class Reference	629
7.376	McPerformanceOption Class Reference	630
7.377	McPricer Class Template Reference	631
7.378	McSimulation Class Template Reference	632
7.379	MCVanillaEngine Class Template Reference	634
7.380	MersenneTwisterUniformRng Class Reference	636
7.381	Merton76Process Class Reference	637
7.382	MixedScheme Class Template Reference	639
7.383	Money Class Reference	641
7.384	MonotonicCubicSpline Class Reference	643
7.385	MonteCarloModel Class Template Reference	644
7.386	MoreGreeks Class Reference	645
7.387	MoroInverseCumulativeNormal Class Reference	646
7.388	MTLCurrency Class Reference	647
7.389	MultiAsset Struct Template Reference	648
7.390	MultiAssetOption Class Reference	649
7.391	MultiAssetOption::arguments Class Reference	651
7.392	MultiAssetOption::results Class Reference	652
7.393	MultiCubicSpline Class Template Reference	653
7.394	MultiPath Class Reference	654
7.395	MultiPathGenerator Class Template Reference	655
7.396	MultiVariate Struct Template Reference	656
7.397	MXNCurrency Class Reference	657
7.398	NaturalCubicSpline Class Reference	658
7.399	NaturalMonotonicCubicSpline Class Reference	659
7.400	NeumannBC Class Reference	660
7.401	Newton Class Reference	662
7.402	NewtonSafe Class Reference	663
7.403	NLGCurrency Class Reference	664
7.404	NoConstraint Class Reference	665

7.405	NOKCurrency Class Reference	666
7.406	NonLinearLeastSquare Class Reference	667
7.407	NormalDistribution Class Reference	668
7.408	NPRCurrency Class Reference	669
7.409	Null Class Template Reference	670
7.410	NullCalendar Class Reference	671
7.411	NullCondition Class Template Reference	672
7.412	NullParameter Class Reference	673
7.413	NumericalMethod Class Reference	674
7.414	NZDCurrency Class Reference	676
7.415	NZDLibor Class Reference	677
7.416	Observable Class Reference	678
7.417	ObservableValue Class Template Reference	679
7.418	Observer Class Reference	680
7.419	OneAssetOption Class Reference	682
7.420	OneAssetOption::arguments Class Reference	685
7.421	OneAssetOption::results Class Reference	686
7.422	OneAssetStrikedOption Class Reference	687
7.423	OneDayCounter Class Reference	689
7.424	OneFactorAffineModel Class Reference	690
7.425	OneFactorModel Class Reference	691
7.426	OneFactorModel::ShortRateDynamics Class Reference	692
7.427	OneFactorModel::ShortRateTree Class Reference	693
7.428	OneFactorOperator Class Reference	694
7.429	OptimizationMethod Class Reference	695
7.430	Option Class Reference	697
7.431	Option::arguments Class Reference	698
7.432	OrnsteinUhlenbeckProcess Class Reference	699
7.433	Oslo Class Reference	701
7.434	Parameter Class Reference	702
7.435	ParameterImpl Class Reference	703
7.436	ParCoupon Class Reference	704
7.437	Path Class Reference	706
7.438	PathGenerator Class Template Reference	707
7.439	PathPricer Class Template Reference	708
7.440	Payoff Class Reference	709

7.441	PercentageStrikePayoff Class Reference	710
7.442	Period Class Reference	711
7.443	PiecewiseConstantParameter Class Reference	712
7.444	PiecewiseYieldCurve Class Template Reference	713
7.445	PKRCurrency Class Reference	715
7.446	PlainVanillaPayoff Class Reference	716
7.447	PLNCurrency Class Reference	717
7.448	PoissonDistribution Class Reference	718
7.449	PositiveConstraint Class Reference	719
7.450	Prague Class Reference	720
7.451	PricingEngine Class Reference	721
7.452	PrimeNumbers Class Reference	722
7.453	Problem Class Reference	723
7.454	PTECurrency Class Reference	724
7.455	QuantoEngine Class Template Reference	725
7.456	QuantoForwardVanillaOption Class Reference	727
7.457	QuantoOptionArguments Class Template Reference	729
7.458	QuantoOptionResults Class Template Reference	730
7.459	QuantoTermStructure Class Reference	731
7.460	QuantoVanillaOption Class Reference	733
7.461	Quote Class Reference	735
7.462	RamdomizedLDS Class Template Reference	736
7.463	RandomSequenceGenerator Class Template Reference	738
7.464	RateHelper Class Reference	739
7.465	Results Class Reference	741
7.466	Ridder Class Reference	742
7.467	Riyadh Class Reference	743
7.468	ROLCurrency Class Reference	744
7.469	Rounding Class Reference	745
7.470	SalvagingAlgorithm Struct Reference	747
7.471	Sample Struct Template Reference	748
7.472	SampledCurve Class Reference	749
7.473	SARCurrency Class Reference	750
7.474	Schedule Class Reference	751
7.475	Secant Class Reference	752
7.476	SeedGenerator Class Reference	753

7.477	SegmentIntegral Class Reference	754
7.478	SEKCurrency Class Reference	755
7.479	Seoul Class Reference	756
7.480	SequenceStatistics Class Template Reference	757
7.481	Settings Class Reference	759
7.482	SGDCurrency Class Reference	761
7.483	Short Class Template Reference	762
7.484	Short< ParCoupon > Class Template Reference	763
7.485	ShortRateModel Class Reference	764
7.486	ShoutCondition Class Reference	766
7.487	SimpleCashFlow Class Reference	767
7.488	SimpleDayCounter Class Reference	768
7.489	SimpleQuote Class Reference	769
7.490	SimpleSwap Class Reference	770
7.491	SimpleSwap::arguments Class Reference	772
7.492	SimpleSwap::results Class Reference	773
7.493	Simplex Class Reference	774
7.494	SimpsonIntegral Class Reference	775
7.495	Singapore Class Reference	776
7.496	SingleAsset Struct Template Reference	777
7.497	SingleAssetOption Class Reference	778
7.498	Singleton Class Template Reference	780
7.499	SingleVariate Struct Template Reference	781
7.500	SITCurrency Class Reference	782
7.501	SKKCurrency Class Reference	783
7.502	SobolRsg Class Reference	784
7.503	Solver1D Class Template Reference	786
7.504	SquareRootProcess Class Reference	788
7.505	StatsHolder Class Reference	789
7.506	SteepestDescent Class Reference	790
7.507	step_iterator Class Template Reference	791
7.508	StepCondition Class Template Reference	792
7.509	StepConditionSet Class Template Reference	793
7.510	StochasticProcess Class Reference	794
7.511	StochasticProcess1D Class Reference	797
7.512	StochasticProcess1D::discretization Class Reference	799

7.513	StochasticProcess::discretization Class Reference	800
7.514	StochasticProcessArray Class Reference	801
7.515	Stock Class Reference	803
7.516	Stockholm Class Reference	804
7.517	StrikedTypePayoff Class Reference	805
7.518	StulzEngine Class Reference	806
7.519	SuperSharePayoff Class Reference	807
7.520	SVD Class Reference	808
7.521	Swap Class Reference	809
7.522	SwapRateHelper Class Reference	811
7.523	Swaption Class Reference	813
7.524	Swaption::arguments Class Reference	815
7.525	Swaption::results Class Reference	816
7.526	SwaptionVolatilityMatrix Class Reference	817
7.527	SwaptionVolatilityStructure Class Reference	819
7.528	Sydney Class Reference	821
7.529	SymmetricSchurDecomposition Class Reference	822
7.530	TabulatedGaussLegendre Class Reference	823
7.531	Taipei Class Reference	824
7.532	Taiwan Class Reference	825
7.533	TARGET Class Reference	826
7.534	TermStructure Class Reference	827
7.535	TermStructureConsistentModel Class Reference	829
7.536	TermStructureFittingParameter Class Reference	830
7.537	THBCurrency Class Reference	831
7.538	Thirty360 Class Reference	832
7.539	Tian Class Reference	833
7.540	Tibor Class Reference	834
7.541	TimeBasket Class Reference	835
7.542	TimeGrid Class Reference	836
7.543	Tokyo Class Reference	838
7.544	Toronto Class Reference	840
7.545	TqrEigenDecomposition Class Reference	841
7.546	TrapezoidIntegral Class Reference	842
7.547	Tree Class Template Reference	844
7.548	TreeCapFloorEngine Class Reference	845

7.549	TreeSwaptionEngine Class Reference	846
7.550	TridiagonalOperator Class Reference	847
7.551	TridiagonalOperator::TimeSetter Class Reference	849
7.552	Trigeorgis Class Reference	850
7.553	TrinomialTree Class Reference	851
7.554	TRLCurrency Class Reference	852
7.555	TRLibor Class Reference	853
7.556	TRYCurrency Class Reference	854
7.557	TTDCurrency Class Reference	855
7.558	TWDCurrency Class Reference	856
7.559	TwoFactorModel Class Reference	857
7.560	TwoFactorModel::ShortRateDynamics Class Reference	858
7.561	TwoFactorModel::ShortRateTree Class Reference	859
7.562	TypePayoff Class Reference	860
7.563	UnitedKingdom Class Reference	861
7.564	UnitedStates Class Reference	863
7.565	UpFrontIndexedCoupon Class Reference	866
7.566	UpRounding Class Reference	867
7.567	USDCurrency Class Reference	868
7.568	USDLibor Class Reference	869
7.569	Value Class Reference	870
7.570	VanillaOption Class Reference	871
7.571	VanillaOption::engine Class Reference	872
7.572	Vasicek Class Reference	873
7.573	Vasicek::Dynamics Class Reference	875
7.574	VEBCurrency Class Reference	876
7.575	Visitor Class Template Reference	877
7.576	Warsaw Class Reference	878
7.577	Wellington Class Reference	879
7.578	Xibor Class Reference	880
7.579	YieldTermStructure Class Reference	882
7.580	ZARCurrency Class Reference	886
7.581	ZeroCouponBond Class Reference	887
7.582	ZeroSpreadedTermStructure Class Reference	888
7.583	ZeroYield Struct Reference	890
7.584	ZeroYieldStructure Class Reference	891

7.585	Zibor Class Reference	893
7.586	Zurich Class Reference	894
8	QuantLib File Documentation	895
8.1	ql/argsandresults.hpp File Reference	895
8.2	ql/calendar.hpp File Reference	897
8.3	ql/Calendars/beijing.hpp File Reference	898
8.4	ql/Calendars/bombay.hpp File Reference	899
8.5	ql/Calendars/bratislava.hpp File Reference	900
8.6	ql/Calendars/budapest.hpp File Reference	901
8.7	ql/Calendars/copenhagen.hpp File Reference	902
8.8	ql/Calendars/germany.hpp File Reference	903
8.9	ql/Calendars/helsinki.hpp File Reference	904
8.10	ql/Calendars/hongkong.hpp File Reference	905
8.11	ql/Calendars/istanbul.hpp File Reference	906
8.12	ql/Calendars/italy.hpp File Reference	907
8.13	ql/Calendars/johannesburg.hpp File Reference	908
8.14	ql/Calendars/jointcalendar.hpp File Reference	909
8.15	ql/Calendars/nullcalendar.hpp File Reference	910
8.16	ql/Calendars/oslo.hpp File Reference	911
8.17	ql/Calendars/prague.hpp File Reference	912
8.18	ql/Calendars/riyadh.hpp File Reference	913
8.19	ql/Calendars/seoul.hpp File Reference	914
8.20	ql/Calendars/singapore.hpp File Reference	915
8.21	ql/Calendars/stockholm.hpp File Reference	916
8.22	ql/Calendars/sydney.hpp File Reference	917
8.23	ql/Calendars/taipei.hpp File Reference	918
8.24	ql/Calendars/taiwan.hpp File Reference	919
8.25	ql/Calendars/target.hpp File Reference	920
8.26	ql/Calendars/tokyo.hpp File Reference	921
8.27	ql/Calendars/toronto.hpp File Reference	922
8.28	ql/Calendars/unitedkingdom.hpp File Reference	923
8.29	ql/Calendars/unitedstates.hpp File Reference	924
8.30	ql/Calendars/warsaw.hpp File Reference	925
8.31	ql/Calendars/wellington.hpp File Reference	926
8.32	ql/Calendars/zurich.hpp File Reference	927
8.33	ql/capvolstructures.hpp File Reference	928

8.34	ql/cashflow.hpp File Reference	929
8.35	ql/CashFlows/analysis.hpp File Reference	930
8.36	ql/CashFlows/basispointsensitivity.hpp File Reference	931
8.37	ql/CashFlows/cashflowvectors.hpp File Reference	932
8.38	ql/CashFlows/coupon.hpp File Reference	933
8.39	ql/CashFlows/fixedratecoupon.hpp File Reference	934
8.40	ql/CashFlows/floatingratecoupon.hpp File Reference	935
8.41	ql/CashFlows/inarrearrindexedcoupon.hpp File Reference	936
8.42	ql/CashFlows/indexedcashflowvectors.hpp File Reference	937
8.43	ql/CashFlows/indexedcoupon.hpp File Reference	938
8.44	ql/CashFlows/parcoupon.hpp File Reference	939
8.45	ql/CashFlows/shortfloatingcoupon.hpp File Reference	940
8.46	ql/CashFlows/shortindexedcoupon.hpp File Reference	941
8.47	ql/CashFlows/simplecashflow.hpp File Reference	942
8.48	ql/CashFlows/timebasket.hpp File Reference	943
8.49	ql/CashFlows/upfrontindexedcoupon.hpp File Reference	944
8.50	ql/Currencies/afrika.hpp File Reference	945
8.51	ql/Currencies/america.hpp File Reference	946
8.52	ql/Currencies/asia.hpp File Reference	948
8.53	ql/Currencies/europe.hpp File Reference	950
8.54	ql/Currencies/exchangeratemanager.hpp File Reference	953
8.55	ql/Currencies/oceania.hpp File Reference	954
8.56	ql/currency.hpp File Reference	955
8.57	ql/date.hpp File Reference	956
8.58	ql/daycounter.hpp File Reference	959
8.59	ql/DayCounters/actual360.hpp File Reference	960
8.60	ql/DayCounters/actual365fixed.hpp File Reference	961
8.61	ql/DayCounters/actualactual.hpp File Reference	962
8.62	ql/DayCounters/one.hpp File Reference	963
8.63	ql/DayCounters/simpliedaycounter.hpp File Reference	964
8.64	ql/DayCounters/thirty360.hpp File Reference	965
8.65	ql/discretizedasset.hpp File Reference	966
8.66	ql/errors.hpp File Reference	967
8.67	ql/exchangerate.hpp File Reference	970
8.68	ql/exercise.hpp File Reference	971
8.69	ql/FiniteDifferences/americancondition.hpp File Reference	972

8.70	ql/FiniteDifferences/boundarycondition.hpp File Reference	973
8.71	ql/FiniteDifferences/bsmoperator.hpp File Reference	974
8.72	ql/FiniteDifferences/bsmtermoperator.hpp File Reference	975
8.73	ql/FiniteDifferences/cranknicolson.hpp File Reference	976
8.74	ql/FiniteDifferences/dminus.hpp File Reference	977
8.75	ql/FiniteDifferences/dplus.hpp File Reference	978
8.76	ql/FiniteDifferences/dplusdminus.hpp File Reference	979
8.77	ql/FiniteDifferences/dzero.hpp File Reference	980
8.78	ql/FiniteDifferences/expliciteuler.hpp File Reference	981
8.79	ql/FiniteDifferences/fdtypedefs.hpp File Reference	982
8.80	ql/FiniteDifferences/finitedifferencemodel.hpp File Reference	983
8.81	ql/FiniteDifferences/impliciteuler.hpp File Reference	984
8.82	ql/FiniteDifferences/mixedscheme.hpp File Reference	985
8.83	ql/FiniteDifferences/onefactoroperator.hpp File Reference	986
8.84	ql/FiniteDifferences/operatortraits.hpp File Reference	987
8.85	ql/FiniteDifferences/parallelevolver.hpp File Reference	988
8.86	ql/FiniteDifferences/shoutcondition.hpp File Reference	989
8.87	ql/FiniteDifferences/stepcondition.hpp File Reference	990
8.88	ql/FiniteDifferences/tridiagonaloperator.hpp File Reference	991
8.89	ql/FiniteDifferences/valueatcenter.hpp File Reference	992
8.90	ql/grid.hpp File Reference	993
8.91	ql/handle.hpp File Reference	994
8.92	ql/history.hpp File Reference	995
8.93	ql/index.hpp File Reference	996
8.94	ql/Indexes/audlibor.hpp File Reference	997
8.95	ql/Indexes/cadlibor.hpp File Reference	998
8.96	ql/Indexes/cdor.hpp File Reference	999
8.97	ql/Indexes/chflibor.hpp File Reference	1000
8.98	ql/Indexes/dkklibor.hpp File Reference	1001
8.99	ql/Indexes/euribor.hpp File Reference	1002
8.100	ql/Indexes/eurlibor.hpp File Reference	1003
8.101	ql/Indexes/gbplibor.hpp File Reference	1004
8.102	ql/Indexes/indexmanager.hpp File Reference	1005
8.103	ql/Indexes/jibor.hpp File Reference	1006
8.104	ql/Indexes/jpylibor.hpp File Reference	1007
8.105	ql/Indexes/libor.hpp File Reference	1008

8.106	ql/Indexes/nzdlbor.hpp File Reference	1009
8.107	ql/Indexes/tibor.hpp File Reference	1010
8.108	ql/Indexes/trlibor.hpp File Reference	1011
8.109	ql/Indexes/usdlbor.hpp File Reference	1012
8.110	ql/Indexes/xibor.hpp File Reference	1013
8.111	ql/Indexes/zibor.hpp File Reference	1014
8.112	ql/instrument.hpp File Reference	1015
8.113	ql/Instruments/asianooption.hpp File Reference	1016
8.114	ql/Instruments/barrierooption.hpp File Reference	1017
8.115	ql/Instruments/basketoption.hpp File Reference	1018
8.116	ql/Instruments/bond.hpp File Reference	1019
8.117	ql/Instruments/callabilityschedule.hpp File Reference	1020
8.118	ql/Instruments/capfloor.hpp File Reference	1021
8.119	ql/Instruments/cliquestoption.hpp File Reference	1022
8.120	ql/Instruments/convertiblebond.hpp File Reference	1023
8.121	ql/Instruments/dividendschedule.hpp File Reference	1024
8.122	ql/Instruments/dividendvanillaoption.hpp File Reference	1025
8.123	ql/Instruments/europeanoption.hpp File Reference	1026
8.124	ql/Instruments/fixedcouponbond.hpp File Reference	1027
8.125	ql/Instruments/floatingratebond.hpp File Reference	1028
8.126	ql/Instruments/forwardvanillaoption.hpp File Reference	1029
8.127	ql/Instruments/multiassetoption.hpp File Reference	1030
8.128	ql/Instruments/oneassetoption.hpp File Reference	1031
8.129	ql/Instruments/oneassetstrikedoption.hpp File Reference	1032
8.130	ql/Instruments/payoffs.hpp File Reference	1033
8.131	ql/Instruments/quantoforwardvanillaoption.hpp File Reference	1034
8.132	ql/Instruments/quantovanillaoption.hpp File Reference	1035
8.133	ql/Instruments/simpleswap.hpp File Reference	1036
8.134	ql/Instruments/stock.hpp File Reference	1037
8.135	ql/Instruments/swap.hpp File Reference	1038
8.136	ql/Instruments/swaption.hpp File Reference	1039
8.137	ql/Instruments/vanillaoption.hpp File Reference	1040
8.138	ql/Instruments/zerocouponbond.hpp File Reference	1041
8.139	ql/interestrates.hpp File Reference	1042
8.140	ql/Lattices/binomialtree.hpp File Reference	1043
8.141	ql/Lattices/bsmllattice.hpp File Reference	1044

8.142	ql/Lattices/lattice.hpp File Reference	1045
8.143	ql/Lattices/lattice1d.hpp File Reference	1046
8.144	ql/Lattices/lattice2d.hpp File Reference	1047
8.145	ql/Lattices/tree.hpp File Reference	1048
8.146	ql/Lattices/trinomialtree.hpp File Reference	1049
8.147	ql/Math/array.hpp File Reference	1050
8.148	ql/Math/backwardflatinterpolation.hpp File Reference	1051
8.149	ql/Math/beta.hpp File Reference	1052
8.150	ql/Math/bicubicsplineinterpolation.hpp File Reference	1053
8.151	ql/Math/bilinearinterpolation.hpp File Reference	1054
8.152	ql/Math/binomialdistribution.hpp File Reference	1055
8.153	ql/Math/bivariatenormaldistribution.hpp File Reference	1056
8.154	ql/Math/chisquaredistribution.hpp File Reference	1057
8.155	ql/Math/choleskydecomposition.hpp File Reference	1058
8.156	ql/Math/comparison.hpp File Reference	1059
8.157	ql/Math/convergencestatistics.hpp File Reference	1060
8.158	ql/Math/cubicspline.hpp File Reference	1061
8.159	ql/Math/discrepancystatistics.hpp File Reference	1062
8.160	ql/Math/errorfunction.hpp File Reference	1063
8.161	ql/Math/extrapolation.hpp File Reference	1064
8.162	ql/Math/factorial.hpp File Reference	1065
8.163	ql/Math/forwardflatinterpolation.hpp File Reference	1066
8.164	ql/Math/functional.hpp File Reference	1067
8.165	ql/Math/gammadistribution.hpp File Reference	1068
8.166	ql/Math/gaussianorthogonalpolynomial.hpp File Reference	1069
8.167	ql/Math/gaussianquadratures.hpp File Reference	1070
8.168	ql/Math/gaussianstatistics.hpp File Reference	1072
8.169	ql/Math/generalstatistics.hpp File Reference	1073
8.170	ql/Math/incompletegamma.hpp File Reference	1074
8.171	ql/Math/incrementalstatistics.hpp File Reference	1075
8.172	ql/Math/interpolation.hpp File Reference	1076
8.173	ql/Math/interpolation2D.hpp File Reference	1077
8.174	ql/Math/kronrodintegral.hpp File Reference	1078
8.175	ql/Math/lexicographicalview.hpp File Reference	1079
8.176	ql/Math/linearinterpolation.hpp File Reference	1080
8.177	ql/Math/loglinearinterpolation.hpp File Reference	1081

8.178	ql/Math/matrix.hpp File Reference	1082
8.179	ql/Math/multicubicspline.hpp File Reference	1083
8.180	ql/Math/normaldistribution.hpp File Reference	1085
8.181	ql/Math/poissondistribution.hpp File Reference	1086
8.182	ql/Math/primenumbers.hpp File Reference	1087
8.183	ql/Math/pseudosqrt.hpp File Reference	1088
8.184	ql/Math/riskstatistics.hpp File Reference	1089
8.185	ql/Math/rounding.hpp File Reference	1090
8.186	ql/Math/sampledcurve.hpp File Reference	1091
8.187	ql/Math/segmentintegral.hpp File Reference	1092
8.188	ql/Math/sequencestatistics.hpp File Reference	1093
8.189	ql/Math/simpsonintegral.hpp File Reference	1095
8.190	ql/Math/statistics.hpp File Reference	1096
8.191	ql/Math/svd.hpp File Reference	1097
8.192	ql/Math/symmetriceigenvalues.hpp File Reference	1098
8.193	ql/Math/symmetricschurdecomposition.hpp File Reference	1099
8.194	ql/Math/tqreigendecomposition.hpp File Reference	1100
8.195	ql/Math/trapezoidintegral.hpp File Reference	1101
8.196	ql/money.hpp File Reference	1102
8.197	ql/MonteCarlo/brownianbridge.hpp File Reference	1103
8.198	ql/MonteCarlo/getcovariance.hpp File Reference	1104
8.199	ql/MonteCarlo/mctraits.hpp File Reference	1105
8.200	ql/MonteCarlo/mctypedefs.hpp File Reference	1106
8.201	ql/MonteCarlo/montecarlomodel.hpp File Reference	1107
8.202	ql/MonteCarlo/multipath.hpp File Reference	1108
8.203	ql/MonteCarlo/multipathgenerator.hpp File Reference	1109
8.204	ql/MonteCarlo/path.hpp File Reference	1110
8.205	ql/MonteCarlo/pathgenerator.hpp File Reference	1111
8.206	ql/MonteCarlo/pathpricer.hpp File Reference	1112
8.207	ql/MonteCarlo/sample.hpp File Reference	1113
8.208	ql/numericalmethod.hpp File Reference	1114
8.209	ql/Optimization/armijo.hpp File Reference	1115
8.210	ql/Optimization/conjugategradient.hpp File Reference	1116
8.211	ql/Optimization/constraint.hpp File Reference	1117
8.212	ql/Optimization/costfunction.hpp File Reference	1118
8.213	ql/Optimization/criteria.hpp File Reference	1119

8.214	ql/Optimization/leastsquare.hpp File Reference	1120
8.215	ql/Optimization/linesearch.hpp File Reference	1121
8.216	ql/Optimization/method.hpp File Reference	1122
8.217	ql/Optimization/problem.hpp File Reference	1123
8.218	ql/Optimization/simplex.hpp File Reference	1124
8.219	ql/Optimization/steepestdescent.hpp File Reference	1125
8.220	ql/option.hpp File Reference	1126
8.221	ql/Patterns/bridge.hpp File Reference	1127
8.222	ql/Patterns/composite.hpp File Reference	1128
8.223	ql/Patterns/curiouslyrecurring.hpp File Reference	1129
8.224	ql/Patterns/lazyobject.hpp File Reference	1130
8.225	ql/Patterns/observable.hpp File Reference	1131
8.226	ql/Patterns/singleton.hpp File Reference	1132
8.227	ql/Patterns/visitor.hpp File Reference	1133
8.228	ql/payoff.hpp File Reference	1134
8.229	ql/Pricers/discretegeometriccaso.hpp File Reference	1135
8.230	ql/Pricers/mccliquestoption.hpp File Reference	1136
8.231	ql/Pricers/mcdiscretearithmeticaso.hpp File Reference	1137
8.232	ql/Pricers/mceverest.hpp File Reference	1138
8.233	ql/Pricers/mchimalaya.hpp File Reference	1139
8.234	ql/Pricers/mcmaxbasket.hpp File Reference	1140
8.235	ql/Pricers/mcpagoda.hpp File Reference	1141
8.236	ql/Pricers/mcperformanceoption.hpp File Reference	1142
8.237	ql/Pricers/mcpricer.hpp File Reference	1143
8.238	ql/Pricers/singleassetoption.hpp File Reference	1144
8.239	ql/pricingengine.hpp File Reference	1145
8.240	ql/PricingEngines/americanpayoffatexpiry.hpp File Reference	1146
8.241	ql/PricingEngines/americanpayoffathit.hpp File Reference	1147
8.242	ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp File Reference	1148
8.243	ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp File Reference . . .	1149
8.244	ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp File Reference	1150
8.245	ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp File Reference	1151
8.246	ql/PricingEngines/Asian/mcdiscreteasianengine.hpp File Reference	1152
8.247	ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference	1153
8.248	ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference	1154
8.249	ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference	1155

8.250	ql/PricingEngines/Basket/mcbasketengine.hpp File Reference	1156
8.251	ql/PricingEngines/Basket/stulzengine.hpp File Reference	1157
8.252	ql/PricingEngines/blackformula.hpp File Reference	1158
8.253	ql/PricingEngines/blackmodel.hpp File Reference	1159
8.254	ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference	1160
8.255	ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference	1161
8.256	ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference	1162
8.257	ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference	1163
8.258	ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference	1164
8.259	ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference	1165
8.260	ql/PricingEngines/Cliquet/mccliquetengine.hpp File Reference	1166
8.261	ql/PricingEngines/Forward/forwardengine.hpp File Reference	1167
8.262	ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference	1168
8.263	ql/PricingEngines/genericmodelengine.hpp File Reference	1169
8.264	ql/PricingEngines/greeks.hpp File Reference	1170
8.265	ql/PricingEngines/latticeshortratemodelengine.hpp File Reference	1171
8.266	ql/PricingEngines/mcsimulation.hpp File Reference	1172
8.267	ql/PricingEngines/Quanto/quantoengine.hpp File Reference	1173
8.268	ql/PricingEngines/Swaption/blackswaptionengine.hpp File Reference	1174
8.269	ql/PricingEngines/Swaption/discretizedswaption.hpp File Reference	1175
8.270	ql/PricingEngines/Swaption/g2swaptionengine.hpp File Reference	1176
8.271	ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp File Reference	1177
8.272	ql/PricingEngines/Swaption/treeswaptionengine.hpp File Reference	1178
8.273	ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference	1179
8.274	ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp File Reference	1180
8.275	ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference	1181
8.276	ql/PricingEngines/Vanilla/analytichestonengine.hpp File Reference	1182
8.277	ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference	1183
8.278	ql/PricingEngines/Vanilla/batesengine.hpp File Reference	1184
8.279	ql/PricingEngines/Vanilla/binomialengine.hpp File Reference	1185
8.280	ql/PricingEngines/Vanilla/bjerkhundstenglandengine.hpp File Reference	1186
8.281	ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference	1187
8.282	ql/PricingEngines/Vanilla/fdamericanengine.hpp File Reference	1188
8.283	ql/PricingEngines/Vanilla/fdbermudanengine.hpp File Reference	1189
8.284	ql/PricingEngines/Vanilla/fddividendamericanengine.hpp File Reference	1190
8.285	ql/PricingEngines/Vanilla/fddividendengine.hpp File Reference	1191

8.286	ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp File Reference	1192
8.287	ql/PricingEngines/Vanilla/fddividendshoutengine.hpp File Reference	1193
8.288	ql/PricingEngines/Vanilla/fdeuropeanengine.hpp File Reference	1194
8.289	ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp File Reference	1195
8.290	ql/PricingEngines/Vanilla/fdshoutengine.hpp File Reference	1196
8.291	ql/PricingEngines/Vanilla/fdstepconditionengine.hpp File Reference	1197
8.292	ql/PricingEngines/Vanilla/fdvanillaengine.hpp File Reference	1198
8.293	ql/PricingEngines/Vanilla/integralengine.hpp File Reference	1199
8.294	ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference	1200
8.295	ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference	1201
8.296	ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference	1202
8.297	ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference	1203
8.298	ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp File Reference	1204
8.299	ql/PricingEngines/Vanilla/mchestonengine.hpp File Reference	1205
8.300	ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference	1206
8.301	ql/Processes/blackscholesprocess.hpp File Reference	1207
8.302	ql/Processes/capletlmmprocess.hpp File Reference	1208
8.303	ql/Processes/defaultable.hpp File Reference	1209
8.304	ql/Processes/eulerdiscretization.hpp File Reference	1210
8.305	ql/Processes/geometricbrownianprocess.hpp File Reference	1211
8.306	ql/Processes/hestonprocess.hpp File Reference	1212
8.307	ql/Processes/merton76process.hpp File Reference	1213
8.308	ql/Processes/ornsteinuhlenbeckprocess.hpp File Reference	1214
8.309	ql/Processes/squarerootprocess.hpp File Reference	1215
8.310	ql/Processes/stochasticprocessarray.hpp File Reference	1216
8.311	ql/qldefines.hpp File Reference	1217
8.312	ql/quote.hpp File Reference	1219
8.313	ql/RandomNumbers/boxmullergaussianrng.hpp File Reference	1220
8.314	ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference	1221
8.315	ql/RandomNumbers/faurersg.hpp File Reference	1222
8.316	ql/RandomNumbers/haltonrng.hpp File Reference	1223
8.317	ql/RandomNumbers/inversecumulativerng.hpp File Reference	1224
8.318	ql/RandomNumbers/inversecumulativersg.hpp File Reference	1225
8.319	ql/RandomNumbers/knuthuniformrng.hpp File Reference	1226
8.320	ql/RandomNumbers/lecuyeruniformrng.hpp File Reference	1227
8.321	ql/RandomNumbers/mt19937uniformrng.hpp File Reference	1228

8.322	ql/RandomNumbers/randomizedlds.hpp File Reference	1229
8.323	ql/RandomNumbers/randomsequencegenerator.hpp File Reference	1230
8.324	ql/RandomNumbers/rngtraits.hpp File Reference	1231
8.325	ql/RandomNumbers/seedgenerator.hpp File Reference	1233
8.326	ql/RandomNumbers/sobolrsg.hpp File Reference	1234
8.327	ql/schedule.hpp File Reference	1235
8.328	ql/settings.hpp File Reference	1236
8.329	ql/ShortRateModels/calibrationhelper.hpp File Reference	1237
8.330	ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference	1238
8.331	ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp File Reference	1239
8.332	ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference	1240
8.333	ql/ShortRateModels/model.hpp File Reference	1241
8.334	ql/ShortRateModels/onefactormodel.hpp File Reference	1242
8.335	ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference	1243
8.336	ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference	1244
8.337	ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference	1245
8.338	ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference	1246
8.339	ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference	1247
8.340	ql/ShortRateModels/parameter.hpp File Reference	1248
8.341	ql/ShortRateModels/twofactormodel.hpp File Reference	1249
8.342	ql/ShortRateModels/TwoFactorModels/batesmodel.hpp File Reference	1250
8.343	ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference	1251
8.344	ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp File Reference	1252
8.345	ql/solver1d.hpp File Reference	1253
8.346	ql/Solvers1D/bisection.hpp File Reference	1254
8.347	ql/Solvers1D/brent.hpp File Reference	1255
8.348	ql/Solvers1D/falseposition.hpp File Reference	1256
8.349	ql/Solvers1D/newton.hpp File Reference	1257
8.350	ql/Solvers1D/newtonsafe.hpp File Reference	1258
8.351	ql/Solvers1D/ridder.hpp File Reference	1259
8.352	ql/Solvers1D/secant.hpp File Reference	1260
8.353	ql/stochasticprocess.hpp File Reference	1261
8.354	ql/swaptionvolstructure.hpp File Reference	1262
8.355	ql/termstructure.hpp File Reference	1263
8.356	ql/TermStructures/affinetermstructure.hpp File Reference	1264
8.357	ql/TermStructures/bondhelpers.hpp File Reference	1265

8.358	ql/TermStructures/bootstraptraits.hpp File Reference	1266
8.359	ql/TermStructures/compoundforward.hpp File Reference	1267
8.360	ql/TermStructures/discountcurve.hpp File Reference	1268
8.361	ql/TermStructures/drifttermstructure.hpp File Reference	1269
8.362	ql/TermStructures/extendeddiscountcurve.hpp File Reference	1270
8.363	ql/TermStructures/flatforward.hpp File Reference	1271
8.364	ql/TermStructures/forwardcurve.hpp File Reference	1272
8.365	ql/TermStructures/forwardspreadedtermstructure.hpp File Reference	1273
8.366	ql/TermStructures/forwardstructure.hpp File Reference	1274
8.367	ql/TermStructures/IMPLIEDtermstructure.hpp File Reference	1275
8.368	ql/TermStructures/piecewiseflatforward.hpp File Reference	1276
8.369	ql/TermStructures/piecewiseyieldcurve.hpp File Reference	1277
8.370	ql/TermStructures/quantotermstructure.hpp File Reference	1278
8.371	ql/TermStructures/ratehelpers.hpp File Reference	1279
8.372	ql/TermStructures/zerocurve.hpp File Reference	1280
8.373	ql/TermStructures/zerospreadedtermstructure.hpp File Reference	1281
8.374	ql/TermStructures/zeroyieldstructure.hpp File Reference	1282
8.375	ql/timegrid.hpp File Reference	1283
8.376	ql/types.hpp File Reference	1284
8.377	ql/Utilities/dataformatters.hpp File Reference	1286
8.378	ql/Utilities/dataparsers.hpp File Reference	1287
8.379	ql/Utilities/disposable.hpp File Reference	1288
8.380	ql/Utilities/null.hpp File Reference	1289
8.381	ql/Utilities/observablevalue.hpp File Reference	1290
8.382	ql/Utilities/steppingiterator.hpp File Reference	1291
8.383	ql/Utilities/strings.hpp File Reference	1292
8.384	ql/Utilities/tracing.hpp File Reference	1293
8.385	ql/Volatilities/blackconstantvol.hpp File Reference	1295
8.386	ql/Volatilities/blackvariancecurve.hpp File Reference	1296
8.387	ql/Volatilities/blackvariancesurface.hpp File Reference	1297
8.388	ql/Volatilities/capflatvolvector.hpp File Reference	1298
8.389	ql/Volatilities/capletconstantvol.hpp File Reference	1299
8.390	ql/Volatilities/capletvariancecurve.hpp File Reference	1300
8.391	ql/Volatilities/IMPLIEDvoltermstructure.hpp File Reference	1301
8.392	ql/Volatilities/localconstantvol.hpp File Reference	1302
8.393	ql/Volatilities/localvolcurve.hpp File Reference	1303

8.394	ql/Volatilities/localvolsurface.hpp File Reference	1304
8.395	ql/Volatilities/swaptionvolmatrix.hpp File Reference	1305
8.396	ql/voltermstructure.hpp File Reference	1306
8.397	ql/yieldtermstructure.hpp File Reference	1307
9	QuantLib Example Documentation	1309
9.1	AmericanOption.cpp	1309
9.2	BermudanSwaption.cpp	1314
9.3	DiscreteHedging.cpp	1319
9.4	EuropeanOption.cpp	1325
9.5	history_iterators.cpp	1331
9.6	swapvaluation.cpp	1332
9.7	tracing_example.cpp	1343
10	Test List	1345
11	Todo List	1357
12	Known Bugs	1361
13	Deprecated List	1363

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 [10](#).

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 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 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.

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 mailing lists

- How do I prevent my auto-responder from posting to the list?

QuantLib cross-platform support

- Does QuantLib support .NET?

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, 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=wl_em_to, and Nando's is at http://www.amazon.co.uk/exec/obidos/registry/Q94W7HUR49Z5/ref=wl_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>) 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>. 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.

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 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.

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 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>](http://sourceforge.net/mailarchive/forum.php?forum=quantlib-users) to catch up on the posts you missed.

1.7.8 QuantLib cross-platform support

Does QuantLib support .NET?

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

1.8 Version history

Release 0.3.11 - October 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 Luttmer. Coefficients for $d \leq 40$ are the same as in Bradley-Fox. For dimension $40 < d \leq 360$ the coefficients have been calculated as optimal values based on the "resolution" criterion. The values has been provided by Christiane Lemieux, private communication, September 2004.
- PathGenerator now works correctly with processes describing S instead of $\log S$. Geometric Brownian process added (thanks to Walter Penschke.)

LATTICE FRAMEWORK

- Reworked the DiscretizedAsset interface.

PRICING ENGINES FRAMEWORK

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

FIXED INCOME

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

YIELD TERM STRUCTURE

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

PATTERNS

- Added Singleton pattern.

MATH

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

MISCELLANEA

- Renamed RelinkableHandle to Handle.

PORTABILITY

- Support for Dev-C++ IDE added.
- Fixes for gcc 2.95 added (thanks to Michael Dirkmann.)

Release 0.3.7 - July 23rd, 2004

IMPORTANT

QuantLib now depends on the Boost library (www.boost.org).

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>. 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 Bjerk Sund 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);
- 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), patch submissions (http://sourceforge.net/tracker/?group_id=12740&atid=312740), and feature requests (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);
- 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, Antoine Cellerier, Aurelien Chanudet, Jon Davidson, Daniele De Francesco, Matteo Gallivanoni, Roman Gitlin, Tomoya Kawanishi, Gary Kennedy, Enrico Michelotti, Gilbert Pepper, Walter Penschke, Gianni Piolanti, Peter Schmitteckert, David Schwartz, Maxim Sokolov, Klaus Spanderen, Marco Tarenghi, Charles Whitmore, Bernd Johannes Wuebben, and Jeff Yu.

1.11 QuantLib License

QuantLib is

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

Copyright (C) 2002, 2003, 2004 Decillion Pty(Ltd)
Copyright (C) 2001, 2002, 2003 Nicolas Di Césaré
Copyright (C) 2003, 2004 Neil Firth
Copyright (C) 2001, 2002, 2003 Sadruddin Rejeb
Copyright (C) 2003 Niels Elken Sønderby

Copyright (C) 2004 FIMAT Group
Copyright (C) 2003, 2004 Roman Gitlin
Copyright (C) 2004 M-Dimension Consulting Inc.
Copyright (C) 2004 Mike Parker
Copyright (C) 2004 Walter Penschke
Copyright (C) 2004 Gianni Piolanti
Copyright (C) 2003 Kawanishi Tomoya
Copyright (C) 2004 Jeff Yu

Copyright (C) 2005 Sercan Atalik
Copyright (C) 2005 Gary Kennedy
Copyright (C) 2004, 2005 Klaus Spanderen
Copyright (C) 2005 Joseph Wang
Copyright (C) 2005 Charles Whitmore

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

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the names of the copyright holders nor the names of the QuantLib Group and its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

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

1.11.1 Comments on Copyright and License

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

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

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

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

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

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

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

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

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

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

Chapter 2

QuantLib Module Index

2.1 QuantLib Modules

Here is a list of all modules:

Numeric types	83
Currencies and FX rates	85
Date and time calculations	89
Calendars	92
Day counters	94
Pricing engines	95
Asian option engines	96
Barrier option engines	97
Basket option engines	98
Cap/floor engines	99
Cliquet option engines	100
Forward option engines	101
Quanto option engines	102
Swaption engines	103
Vanilla option engines	104
Finite-differences framework	106
Short-rate modelling framework	112
Financial instruments	115
Lattice methods	118
Math tools	121
Monte Carlo framework	123
Design patterns	129
Term structures	130
Utilities	132
QuantLib macros	134
Generic macros	135
Numeric limits	136
Template capabilities	137
Iterator support	138
Debugging macros	140
Output manipulators	139

Chapter 3

QuantLib Hierarchical Index

3.1 QuantLib Class Hierarchy

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

AcyclicVisitor	146
BPSBasketCalculator	239
BPSCalculator	240
AmericanPayoffAtExpiry	153
AmericanPayoffAtHit	154
Arguments	165
CapFloor::arguments	264
Option::arguments	698
MultiAssetOption::arguments	651
BasketOption::arguments	186
OneAssetOption::arguments	685
Swaption::arguments	815
SimpleSwap::arguments	772
Swaption::arguments	815
Array	167
Average	175
BackwardFlat	176
Barrier	179
BarrierOption::arguments	182
Bicubic	196
Bilinear	198
BinomialDistribution	200
BivariateCumulativeNormalDistributionDr78	204
BivariateCumulativeNormalDistributionWe04DP	205
BlackFormula	210
BlackKarasinski::Dynamics	213
BoundaryCondition	235
BoundaryCondition< TridiagonalOperator >	235
DirichletBC	347
NeumannBC	660
BoxMullerGaussianRng	238
Bridge	243

Bridge< Calendar, CalendarImpl >	243
Calendar	252
Beijing	193
Bombay	231
Bratislava	241
Budapest	248
Copenhagen	312
Germany	485
Helsinki	492
HongKong	503
Istanbul	553
Italy	554
Johannesburg	560
JointCalendar	561
NullCalendar	671
Oslo	701
Prague	720
Riyadh	743
Seoul	756
Singapore	776
Stockholm	804
Sydney	821
Taipei	824
Taiwan	825
TARGET	826
Tokyo	838
Toronto	840
UnitedKingdom	861
UnitedStates	863
Warsaw	878
Wellington	879
Zurich	894
Bridge< Constraint, ConstraintImpl >	243
Constraint	300
BoundaryConstraint	237
CompositeConstraint	294
NoConstraint	665
PositiveConstraint	719
Bridge< DayCounter, DayCounterImpl >	243
DayCounter	340
Actual360	143
Actual365Fixed	144
ActualActual	145
OneDayCounter	689
SimpleDayCounter	768
Thirty360	832
Bridge< Interpolation, InterpolationImpl >	243
Interpolation	540
BackwardFlatInterpolation	177
CubicSpline	324
MonotonicCubicSpline	643
NaturalCubicSpline	658

NaturalMonotonicCubicSpline	659
ForwardFlatInterpolation	437
LinearInterpolation	585
LogLinearInterpolation	598
Bridge< Interpolation2D, Interpolation2DImpl >	243
Interpolation2D	541
BicubicSpline	197
BilinearInterpolation	199
Bridge< Parameter, ParameterImpl >	243
Parameter	702
ConstantParameter	299
NullParameter	673
PiecewiseConstantParameter	712
TermStructureFittingParameter	830
ExtendedCoxIngersollRoss::FittingParameter	403
G2::FittingParameter	453
HullWhite::FittingParameter	508
BrownianBridge	245
CalendarImpl	258
Calendar::WesternImpl	257
Cashflows	277
CLGaussianRng	284
CliquetOption::arguments	287
Composite	293
ConstraintImpl	301
ContinuousAveragingAsianOption::arguments	304
ConvergenceStatistics	306
ConvertibleBond::option::arguments	309
CostFunction	313
LeastSquareFunction	577
CovarianceDecomposition	316
CoxIngersollRoss::Dynamics	319
ExtendedCoxIngersollRoss::Dynamics	402
Cubic	323
CumulativeBinomialDistribution	326
CumulativeNormalDistribution	327
CumulativePoissonDistribution	328
CuriouslyRecurringTemplate	329
Lattice	570
Lattice1D	572
Lattice2D	573
Solver1D	786
CuriouslyRecurringTemplate< AdditiveEQPBinoialTree >	329
Tree< AdditiveEQPBinoialTree >	844
BinomialTree< AdditiveEQPBinoialTree >	201
EqualProbabilitiesBinomialTree< AdditiveEQPBinoialTree >	382
AdditiveEQPBinoialTree	147
CuriouslyRecurringTemplate< Bisection >	329
Solver1D< Bisection >	786
Bisection	203

CuriouslyRecurringTemplate< BlackScholesLattice< T > >	329
Lattice< BlackScholesLattice< T > >	570
Lattice1D< BlackScholesLattice< T > >	572
BlackScholesLattice	216
CuriouslyRecurringTemplate< Brent >	329
Solver1D< Brent >	786
Brent	242
CuriouslyRecurringTemplate< CoxRossRubinstein >	329
Tree< CoxRossRubinstein >	844
BinomialTree< CoxRossRubinstein >	201
EqualJumpsBinomialTree< CoxRossRubinstein >	381
CoxRossRubinstein	320
CuriouslyRecurringTemplate< FalsePosition >	329
Solver1D< FalsePosition >	786
FalsePosition	408
CuriouslyRecurringTemplate< JarrowRudd >	329
Tree< JarrowRudd >	844
BinomialTree< JarrowRudd >	201
EqualProbabilitiesBinomialTree< JarrowRudd >	382
JarrowRudd	558
CuriouslyRecurringTemplate< LeisenReimer >	329
Tree< LeisenReimer >	844
BinomialTree< LeisenReimer >	201
LeisenReimer	580
CuriouslyRecurringTemplate< Newton >	329
Solver1D< Newton >	786
Newton	662
CuriouslyRecurringTemplate< NewtonSafe >	329
Solver1D< NewtonSafe >	786
NewtonSafe	663
CuriouslyRecurringTemplate< OneFactorModel::ShortRateTree >	329
Lattice< OneFactorModel::ShortRateTree >	570
Lattice1D< OneFactorModel::ShortRateTree >	572
OneFactorModel::ShortRateTree	693
CuriouslyRecurringTemplate< Ridder >	329
Solver1D< Ridder >	786
Ridder	742
CuriouslyRecurringTemplate< Secant >	329
Solver1D< Secant >	786
Secant	752
CuriouslyRecurringTemplate< T >	329
Tree	844
BinomialTree	201
EqualJumpsBinomialTree	381
EqualProbabilitiesBinomialTree	382
CuriouslyRecurringTemplate< Tian >	329
Tree< Tian >	844
BinomialTree< Tian >	201

Tian	833
CuriouslyRecurringTemplate< Trigeorgis >	329
Tree< Trigeorgis >	844
BinomialTree< Trigeorgis >	201
EqualJumpsBinomialTree< Trigeorgis >	381
Trigeorgis	850
CuriouslyRecurringTemplate< TrinomialTree >	329
Tree< TrinomialTree >	844
TrinomialTree	851
CuriouslyRecurringTemplate< TwoFactorModel::ShortRateTree >	329
Lattice< TwoFactorModel::ShortRateTree >	570
Lattice2D< TwoFactorModel::ShortRateTree, TrinomialTree >	573
TwoFactorModel::ShortRateTree	859
Currency	330
ARSCurrency	170
ATSCurrency	172
AUDCurrency	173
BDTCurrency	191
BEFCurrency	192
BGLCurrency	195
BRLCurrency	244
BYRCurrency	249
CADCurrency	250
CHFCurrency	282
CLPCurrency	290
CNYCurrency	291
COPCurrency	311
CYPCurrency	334
CZKCurrency	335
DEMCurrency	343
DKKCurrency	367
EEKCurrency	378
ESPCurrency	385
EURCurrency	388
FIMCurrency	421
FRFCurrency	449
GBPCurrency	474
GRDCurrency	488
HKDCurrency	502
HUFCurrency	504
IEPCurrency	509
ILSCurrency	510
INRCurrency	526
IQDCurrency	550
IRRCurrency	551
ISKCurrency	552
ITLCurrency	556
JPYCurrency	562
KRWCurrency	568
KWDCurrency	569
LTLCurrency	599
LUFCurrency	600

LVLCurrency	601
MTLCurrency	647
MXNCurrency	657
NLGCurrency	664
NOKCurrency	666
NPRCurrency	669
NZDCurrency	676
PKRCurrency	715
PLNCurrency	717
PTECurrency	724
ROLCurrency	744
SARCurrency	750
SEKCurrency	755
SGDCurrency	761
SITCurrency	782
SKKCurrency	783
THBCurrency	831
TRLCurrency	852
TRYCurrency	854
TTDCurrency	855
TWDCurrency	856
USDCurrency	868
VEBCurrency	876
ZARCurrency	886
Date	336
DayCounterImpl	342
Discount	349
DiscreteAveragingAsianOption::arguments	353
DiscretizedAsset	356
DiscretizedDiscountBond	359
DiscretizedOption	360
Disposable	362
DividendVanillaOption::arguments	365
Duration	375
EndCriteria	379
ErrorFunction	384
exception	
Error	383
ExchangeRate	393
Exercise	397
EarlyExercise	377
AmericanExercise	152
BermudanExercise	194
EuropeanExercise	391
Extrapolator	406
BlackVolTermStructure	228
BlackVarianceTermStructure	224
BlackVarianceCurve	220
BlackVarianceSurface	222
ImpliedVolTermStructure	515
BlackVolatilityTermStructure	226
BlackConstantVol	208

CapletVolatilityStructure	270
CapletConstantVolatility	266
CapVolatilityStructure	272
CapVolatilityVector	274
LocalVolTermStructure	595
LocalConstantVol	590
LocalVolCurve	591
LocalVolSurface	593
SwaptionVolatilityStructure	819
SwaptionVolatilityMatrix	817
YieldTermStructure	882
AffineTermStructure	149
FlatForward	428
ForwardRateStructure	441
CompoundForward	296
ForwardSpreadedTermStructure	443
InterpolatedForwardCurve	536
ImpliedTermStructure	513
InterpolatedDiscountCurve	534
ExtendedDiscountCurve	404
InterpolatedDiscountCurve< LogLinear >	534
ZeroYieldStructure	891
DriftTermStructure	373
InterpolatedZeroCurve	538
QuantoTermStructure	731
ZeroSpreadedTermStructure	888
Factorial	407
FaureRsg	409
FDDividendEngine	413
FDDividendAmericanEngine	412
FDDividendEuropeanEngine	414
FDDividendShoutEngine	415
FDVanillaEngine	419
FDEuropeanEngine	416
FDStepConditionEngine	418
FDAmericanEngine	410
FDShoutEngine	417
FiniteDifferenceModel	422
ForwardFlat	436
ForwardOptionArguments	438
ForwardRate	440
GammaFunction	455
GaussianOrthogonalPolynomial	464
GaussHermitePolynomial	461
GaussHyperbolicPolynomial	463
GaussJacobiPolynomial	470
GaussLaguerrePolynomial	472
GaussianQuadrature	465
GaussChebyshev2thIntegration	457
GaussChebyshevIntegration	458
GaussGegenbauerIntegration	459

GaussHermiteIntegration	460
GaussHyperbolicIntegration	462
GaussJacobiIntegration	469
GaussLaguerreIntegration	471
GaussLegendreIntegration	473
GaussianStatistics	466
GeneralStatistics	476
GenericRiskStatistics	481
HaltonRsg	490
Handle	491
History	497
History::const_iterator	500
History::Entry	501
HullWhite::Dynamics	507
IMM	511
IncrementalStatistics	519
InterestRate	531
Interpolation2DImpl	543
Interpolation2D::templateImpl	542
InterpolationImpl	545
Interpolation::templateImpl	544
InverseCumulativeNormal	546
InverseCumulativePoisson	547
InverseCumulativeRng	548
InverseCumulativeRsg	549
KnuthUniformRng	566
KronrodIntegral	567
LeastSquareProblem	578
LecuyerUniformRng	579
LexicographicalView	581
Linear	584
LineSearch	586
ArmijoLineSearch	166
LogLinear	597
MakeMCDigitalEngine	602
MakeMCEuropeanEngine	603
MakeMCEuropeanHestonEngine	604
MakeSchedule	605
map< Date, Real >	
TimeBasket	835
Matrix	606
McPricer	631
McPricer< MultiVariate< PseudoRandom > >	631
McEverest	625
McHimalaya	627
McMaxBasket	628
McPagoda	629
McPricer< SingleVariate< PseudoRandom > >	631
McCliquetOption	615
McDiscreteArithmeticASO	619
McPerformanceOption	630
McSimulation	632

McSimulation< MultiVariate< RNG >, S >	632
MCBasketEngine	613
MCHestonEngine	626
MCEuropeanHestonEngine	624
McSimulation< SingleVariate< RNG >, S >	632
MCBarrierEngine	611
MCDiscreteAveragingAsianEngine	620
MCDiscreteArithmeticAPEngine	617
MCDiscreteGeometricAPEngine	622
MCVanillaEngine	634
MCDigitalEngine	616
MCEuropeanEngine	623
MersenneTwisterUniformRng	636
MixedScheme	639
CrankNicolson	321
ExplicitEuler	398
ImplicitEuler	512
Money	641
MonteCarloModel	644
MoroInverseCumulativeNormal	646
MultiAsset	648
MultiCubicSpline	653
MultiPath	654
MultiPathGenerator	655
MultiVariate	656
NonLinearLeastSquare	667
NormalDistribution	668
Null	670
NumericalMethod	674
Lattice	570
Lattice< BlackScholesLattice< T > >	570
Lattice< OneFactorModel::ShortRateTree >	570
Lattice< TwoFactorModel::ShortRateTree >	570
Observable	678
AffineModel	148
G2	451
OneFactorAffineModel	690
CoxIngersollRoss	317
ExtendedCoxIngersollRoss	400
Vasicek	873
HullWhite	505
BlackModel	214
CalibrationHelper	259
HestonModelHelper	494
CashFlow	276
Coupon	314
FixedRateCoupon	426
FloatingRateCoupon	431
IndexedCoupon	523
InArrearIndexedCoupon	517
UpFrontIndexedCoupon	866

ParCoupon	704
Short< ParCoupon >	763
SimpleCashFlow	767
Index	522
Xibor	880
Cdor	280
Euribor	389
Jibar	559
Libor	583
AUDLibor	174
CADLibor	251
CHFLibor	283
DKKLibor	368
EURLibor	390
GBPLibor	475
JPYLibor	563
NZDLibor	677
USDLibor	869
Tibor	834
TRLibor	853
Zibor	893
LazyObject	575
AffineTermStructure	149
Instrument	527
Bond	232
FixedCouponBond	423
FloatingRateBond	430
ZeroCouponBond	887
CapFloor	262
Cap	261
Collar	292
Floor	433
Option	697
MultiAssetOption	649
BasketOption	184
OneAssetOption	682
OneAssetStrikedOption	687
BarrierOption	180
CliquetOption	285
ContinuousAveragingAsianOption	302
ConvertibleBond::option	307
DiscreteAveragingAsianOption	351
VanillaOption	871
DividendVanillaOption	363
EuropeanOption	392
ForwardVanillaOption	445
QuantoVanillaOption	733
QuantoForwardVanillaOption	727
Swaption	813
Stock	803
Swap	809
SimpleSwap	770

PiecewiseYieldCurve	713
Link	588
PricingEngine	721
GenericEngine	479
GenericModelEngine	480
GenericEngine< Arguments, Results >	479
GenericModelEngine< ShortRateModel, Arguments, Results >	480
LatticeShortRateModelEngine	574
GenericEngine< BarrierOption::arguments, BarrierOption::results >	479
BarrierOption::engine	183
AnalyticBarrierEngine	155
MCBarrierEngine	611
GenericEngine< BasketOption::arguments, BasketOption::results >	479
BasketOption::engine	187
MCAmericanBasketEngine	610
MCBasketEngine	613
StulzEngine	806
GenericEngine< CapFloor::arguments, CapFloor::results >	479
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >	480
AnalyticCapFloorEngine	156
GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results >	480
BlackCapFloorEngine	207
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >	480
LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >	574
TreeCapFloorEngine	845
GenericEngine< CliquetOption::arguments, CliquetOption::results >	479
CliquetOption::engine	288
AnalyticCliquetEngine	157
AnalyticPerformanceEngine	164
GenericEngine< ContinuousAveragingAsianOption::arguments, ContinuousAveragingAsianOption::results >	479
ContinuousAveragingAsianOption::engine	305
AnalyticContinuousGeometricAveragePriceAsianEngine	158
GenericEngine< ConvertibleBond::option::arguments, ConvertibleBond::option::results >	479
ConvertibleBond::option::engine	310
GenericEngine< DiscreteAveragingAsianOption::arguments, DiscreteAveragingAsianOption::results >	479
DiscreteAveragingAsianOption::engine	354
AnalyticDiscreteGeometricAveragePriceAsianEngine	160
MCDiscreteAveragingAsianEngine	620
GenericEngine< DividendVanillaOption::arguments, DividendVanillaOption::results >	479
DividendVanillaOption::engine	366
AnalyticDividendEuropeanEngine	161
FDBermudanEngine	411
FDDividendAmericanEngine	412
FDDividendEuropeanEngine	414
FDDividendShoutEngine	415
GenericEngine< ForwardOptionArguments< ArgumentsType >, ResultsType >	479

ForwardEngine	435
ForwardPerformanceEngine	439
GenericEngine< OneAssetOption::arguments, OneAssetOption::results >	479
GenericEngine< QuantoOptionArguments< ArgumentsType >, QuantoOption- Results< ResultsType > >	479
QuantoEngine	725
GenericEngine< Swaption::arguments, Swaption::results >	479
GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results >	480
BlackSwaptionEngine	219
GenericModelEngine< G2, Swaption::arguments, Swaption::results >	480
G2SwaptionEngine	454
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swap- tion::results >	480
JamshidianSwaptionEngine	557
GenericModelEngine< ShortRateModel, Swaption::arguments, Swap- tion::results >	480
LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >	574
TreeSwaptionEngine	846
GenericEngine< VanillaOption::arguments, VanillaOption::results >	479
GenericModelEngine< HestonModel, VanillaOption::arguments, Vanilla- Option::results >	480
AnalyticHestonEngine	163
BatesEngine	188
Quote	735
CompositeQuote	295
DerivedQuote	346
SimpleQuote	769
RateHelper	739
DepositRateHelper	344
FixedCouponBondHelper	424
FraRateHelper	447
FuturesRateHelper	450
SwapRateHelper	811
ShortRateModel	764
HestonModel	493
BatesModel	190
OneFactorModel	691
BlackKarasinski	212
OneFactorAffineModel	690
TwoFactorModel	857
G2	451
StochasticProcess	794
CapletLiborMarketModelProcess	268
HestonProcess	495
StochasticProcess1D	797
BlackScholesProcess	217
GeometricBrownianMotionProcess	484
Merton76Process	637
OrnsteinUhlenbeckProcess	699
SquareRootProcess	788
StochasticProcessArray	801

TermStructure	827
BlackVolTermStructure	228
CapletVolatilityStructure	270
CapVolatilityStructure	272
LocalVolTermStructure	595
SwaptionVolatilityStructure	819
YieldTermStructure	882
TermStructureConsistentModel	829
BlackKarasinski	212
ExtendedCoxIngersollRoss	400
G2	451
HullWhite	505
ObservableValue	679
ObservableValue< Date >	679
Observer	680
BlackModel	214
CalibrationHelper	259
CompositeQuote	295
DerivedQuote	346
GenericModelEngine	480
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results > . . .	480
GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results > . . .	480
GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results > . . .	480
GenericModelEngine< G2, Swaption::arguments, Swaption::results >	480
GenericModelEngine< HestonModel, VanillaOption::arguments, VanillaOption::results >	480
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results >	480
GenericModelEngine< ShortRateModel, Arguments, Results >	480
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results > .	480
GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results > .	480
IndexedCoupon	523
LazyObject	575
Link	588
ParCoupon	704
RateHelper	739
ShortRateModel	764
StochasticProcess	794
TermStructure	827
Xibor	880
OneFactorModel::ShortRateDynamics	692
OptimizationMethod	695
ConjugateGradient	298
Simplex	774
SteepestDescent	790
ParameterImpl	703
Path	706
PathGenerator	707
PathPricer	708
PathPricer< MultiPath >	708
PathPricer< Path >	708
Payoff	709
TypePayoff	860

StrikedTypePayoff	805
AssetOrNothingPayoff	171
CashOrNothingPayoff	279
GapPayoff	456
PercentageStrikePayoff	710
PlainVanillaPayoff	716
SuperSharePayoff	807
Period	711
PoissonDistribution	718
PrimeNumbers	722
Problem	723
QuantoOptionArguments	729
QuantoOptionResults	730
RandomizedLDS	736
RandomSequenceGenerator	738
Results	741
Greeks	489
MultiAssetOption::results	652
OneAssetOption::results	686
MoreGreeks	645
OneAssetOption::results	686
Value	870
CapFloor::results	265
MultiAssetOption::results	652
OneAssetOption::results	686
SimpleSwap::results	773
Swaption::results	816
Rounding	745
CeilingTruncation	281
ClosestRounding	289
DownRounding	370
FloorTruncation	434
UpRounding	867
SalvagingAlgorithm	747
Sample	748
SampledCurve	749
Schedule	751
SegmentIntegral	754
SequenceStatistics	757
SequenceStatistics< Statistics >	757
DiscrepancyStatistics	350
Short	762
SingleAsset	777
SingleAssetOption	778
DiscreteGeometricASO	355
Singleton	780
ExchangeRateManager	395
IndexManager	525
SeedGenerator	753
Settings	759
Singleton< ExchangeRateManager >	780
Singleton< IndexManager >	780

Singleton< SeedGenerator >	780
Singleton< Settings >	780
Singleton< Tracing >	780
SingleVariate	781
SobolRsg	784
StatsHolder	789
step_iterator	791
StepCondition	792
AmericanCondition	151
NullCondition	672
ShoutCondition	766
StepCondition< Array >	792
StepConditionSet	793
StochasticProcess1D::discretization	799
EulerDiscretization	386
StochasticProcess::discretization	800
EulerDiscretization	386
SVD	808
SymmetricSchurDecomposition	822
TabulatedGaussLegendre	823
TimeGrid	836
TqrEigenDecomposition	841
TrapezoidIntegral	842
SimpsonIntegral	775
TridiagonalOperator	847
BSMOperator	246
BSMTermOperator	247
DMinus	369
DPlus	371
DPlusDMinus	372
DZero	376
OneFactorOperator	694
TridiagonalOperator::TimeSetter	849
TwoFactorModel::ShortRateDynamics	858
VanillaOption::engine	872
AnalyticDigitalAmericanEngine	159
AnalyticEuropeanEngine	162
BaroneAdesiWhaleyApproximationEngine	178
BinomialVanillaEngine	202
Bjerk sundStenslandApproximationEngine	206
FDShoutEngine	417
IntegralEngine	530
JumpDiffusionEngine	564
JuQuadraticApproximationEngine	565
MCHestonEngine	626
MCVanillaEngine	634
Vasicek::Dynamics	875
Visitor	877
Visitor< CashFlow >	877
BPSBasketCalculator	239
BPSCalculator	240
Visitor< Coupon >	877

BPSBasketCalculator	239
BPSCalculator	240
Visitor< FixedRateCoupon >	877
BPSBasketCalculator	239
ZeroYield	890

Chapter 4

QuantLib Class Index

4.1 QuantLib Class List

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

Actual360 (Actual/360 day count convention)	143
Actual365Fixed (Actual/365 (Fixed) day count convention)	144
ActualActual (Actual/Actual day count)	145
AcyclicVisitor (Degenerate base class for the Acyclic Visitor pattern)	146
AdditiveEQPBinomialTree (Additive equal probabilities binomial tree)	147
AffineModel (Affine model class)	148
AffineTermStructure (Term-structure implied by an affine model)	149
AmericanCondition (American exercise condition)	151
AmericanExercise (American exercise)	152
AmericanPayoffAtExpiry	153
AmericanPayoffAtHit	154
AnalyticBarrierEngine (Pricing engine for barrier options using analytical formulae) . .	155
AnalyticCapFloorEngine (Analytic engine for cap/floor)	156
AnalyticCliquetEngine (Pricing engine for Cliquet options using analytical formulae) .	157
AnalyticContinuousGeometricAveragePriceAsianEngine (Pricing engine for European continuous geometric average price Asian)	158
AnalyticDigitalAmericanEngine	159
AnalyticDiscreteGeometricAveragePriceAsianEngine (Pricing engine for European dis- crete geometric average price Asian)	160
AnalyticDividendEuropeanEngine (Analytic pricing engine for European options with discrete dividends)	161
AnalyticEuropeanEngine (Pricing engine for European vanilla options using analytical formulae)	162
AnalyticHestonEngine (Analytic Heston-model engine based on Fourier transform) . .	163
AnalyticPerformanceEngine (Pricing engine for performance options using analytical formulae)	164
Arguments (Base class for generic argument groups)	165
ArmijoLineSearch (Armijo line search)	166
Array (1-D array used in linear algebra)	167
ARSCurrency (Argentinian peso)	170
AssetOrNothingPayoff (Binary asset-or-nothing payoff)	171
ATSCurrency (Austrian shilling)	172
AUDCurrency (Australian dollar)	173

AUDLibor (AUD LIBOR rate)	174
Average (Placeholder for enumerated averaging types)	175
BackwardFlat (Backward-flat interpolation factory and traits)	176
BackwardFlatInterpolation (Backward-flat interpolation between discrete points)	177
BaroneAdesiWhaleyApproximationEngine	178
Barrier (Placeholder for enumerated barrier types)	179
BarrierOption (Barrier option on a single asset)	180
BarrierOption::arguments (Arguments for barrier option calculation)	182
BarrierOption::engine (Barrier engine base class)	183
BasketOption (Basket option on a number of assets)	184
BasketOption::arguments (Arguments for basket option calculation)	186
BasketOption::engine (Basket option engine base class)	187
BatesEngine (Bates model engines based on Fourier transform)	188
BatesModel	190
BDTCurrency (Bangladesh taka)	191
BEFCurrency (Belgian franc)	192
Beijing (Beijing calendar)	193
BermudanExercise (Bermudan exercise)	194
BGLCurrency (Bulgarian lev)	195
Bicubic (Bicubic-spline interpolation factory)	196
BicubicSpline	197
Bilinear (Bilinear interpolation factory)	198
BilinearInterpolation (Bilinear interpolation between discrete points)	199
BinomialDistribution (Binomial probability distribution function)	200
BinomialTree (Binomial tree base class)	201
BinomialVanillaEngine (Pricing engine for vanilla options using binomial trees)	202
Bisection (Bisection 1-D solver)	203
BivariateCumulativeNormalDistributionDr78 (Cumulative bivariate normal distribution function)	204
BivariateCumulativeNormalDistributionWe04DP (Cumulative bivariate normal distribution function (West 2004))	205
BjerkstrandStenslandApproximationEngine	206
BlackCapFloorEngine (Black-formula cap/floor engine)	207
BlackConstantVol (Constant Black volatility, no time-strike dependence)	208
BlackFormula (Black-formula calculator)	210
BlackKarasinski (Standard Black-Karasinski model class)	212
BlackKarasinski::Dynamics (Short-rate dynamics in the Black-Karasinski model)	213
BlackModel (Black-model for vanilla interest-rate derivatives)	214
BlackScholesLattice (Simple binomial lattice approximating the Black-Scholes model)	216
BlackScholesProcess (Black-Scholes stochastic process)	217
BlackSwaptionEngine (Black-formula swaption engine)	219
BlackVarianceCurve (Black volatility swaption modelled as variance curve)	220
BlackVarianceSurface (Black volatility surface modelled as variance surface)	222
BlackVarianceTermStructure (Black variance term structure)	224
BlackVolatilityTermStructure (Black-volatility term structure)	226
BlackVolTermStructure (Black-volatility term structure)	228
Bombay (Bombay calendar)	231
Bond (Base bond class)	232
BoundaryCondition (Abstract boundary condition class for finite difference problems)	235
BoundaryConstraint (Constraint imposing all arguments to be in [low,high])	237
BoxMullerGaussianRng (Gaussian random number generator)	238
BPSBasketCalculator	239
BPSCalculator (Basis point sensitivity (BPS) calculator)	240
Bratislava (Bratislava calendar)	241

Brent (Brent 1-D solver)	242
Bridge (The Bridge pattern made explicit)	243
BRLCurrency (Brazilian real)	244
BrownianBridge (Builds Wiener process paths using Gaussian variates)	245
BSMOperator (Black-Scholes-Merton differential operator)	246
BSMTermOperator (Black-Scholes-Merton differential operator)	247
Budapest (Budapest calendar)	248
BYRCurrency (Belarussian ruble)	249
CADCurrency (Canadian dollar)	250
CADLibor (CAD LIBOR rate)	251
Calendar (calendar class)	252
Calendar::WesternImpl (Partial calendar implementation)	257
CalendarImpl (Abstract base class for calendar implementations)	258
CalibrationHelper (Liquid market instrument used during calibration)	259
Cap (Concrete cap class)	261
CapFloor (Base class for cap-like instruments)	262
CapFloor::arguments (Arguments for cap/floor calculation)	264
CapFloor::results (Results from cap/floor calculation)	265
CapletConstantVolatility (Constant caplet volatility, no time-strike dependence)	266
CapletLiborMarketModelProcess (Caplet libor-market-model process)	268
CapletVolatilityStructure (Caplet/floorlet forward-volatility structure)	270
CapVolatilityStructure (Cap/floor term-volatility structure)	272
CapVolatilityVector (Cap/floor at-the-money term-volatility vector)	274
CashFlow (Base class for cash flows)	276
Cashflows (Cashflows analysis functions)	277
CashOrNothingPayoff (Binary cash-or-nothing payoff)	279
Cdor (CDOR rate)	280
CeilingTruncation (Ceiling truncation)	281
CHFCurrency (Swiss franc)	282
CHFLibor (CHF LIBOR rate)	283
CLGaussianRng (Gaussian random number generator)	284
CliquetOption (Cliquet (Ratchet) option)	285
CliquetOption::arguments (Arguments for cliquet option calculation)	287
CliquetOption::engine (Cliquet engine base class)	288
ClosestRounding (Closest rounding)	289
CLPCurrency (Chilean peso)	290
CNYCurrency (Chinese yuan)	291
Collar (Concrete collar class)	292
Composite (Composite pattern)	293
CompositeConstraint (Constraint enforcing both given sub-constraints)	294
CompositeQuote (Market element whose value depends on two other market element)	295
CompoundForward (Compound-forward structure)	296
ConjugateGradient (Multi-dimensional Conjugate Gradient class)	298
ConstantParameter (Standard constant parameter $a(t) = a$)	299
Constraint (Base constraint class)	300
ConstraintImpl (Base class for constraint implementations)	301
ContinuousAveragingAsianOption (Continuous-averaging Asian option)	302
ContinuousAveragingAsianOption::arguments (Extra arguments for single-asset continuous-average Asian option)	304
ContinuousAveragingAsianOption::engine (Continuous-averaging Asian engine base class)	305
ConvergenceStatistics (Statistics class with convergence table)	306
ConvertibleBond::option (Option like features for Convertible Bond calculation)	307
ConvertibleBond::option::arguments (Arguments for Convertible Bond calculation)	309

ConvertibleBond::option::engine (Convertible bond engine base class)	310
COPCurrency (Colombian peso)	311
Copenhagen (Copenhagen calendar)	312
CostFunction (Cost function abstract class for optimization problem)	313
Coupon (coupon accruing over a fixed period)	314
CovarianceDecomposition	316
CoxIngersollRoss (Cox-Ingersoll-Ross model class)	317
CoxIngersollRoss::Dynamics (Dynamics of the short-rate under the Cox-Ingersoll-Ross model)	319
CoxRossRubinstein (Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree)	320
CrankNicolson (Crank-Nicolson scheme for finite difference methods)	321
Cubic (cubic-spline interpolation factory and traits)	323
CubicSpline (Cubic spline interpolation between discrete points)	324
CumulativeBinomialDistribution (Cumulative binomial distribution function)	326
CumulativeNormalDistribution (Cumulative normal distribution function)	327
CumulativePoissonDistribution (Cumulative Poisson distribution function)	328
CuriouslyRecurringTemplate (Support for the curiously recurring template pattern)	329
Currency (Currency specification)	330
CYPCurrency (Cyprus pound)	334
CZKCurrency (Czech koruna)	335
Date (Concrete date class)	336
DayCounter (Day counter class)	340
DayCounterImpl (Abstract base class for day counter implementations)	342
DEMCurrency (Deutsche mark)	343
DepositRateHelper (Deposit rate)	344
DerivedQuote (Market element whose value depends on another market element)	346
DirichletBC (Neumann boundary condition (i.e., constant value))	347
Discount (Discount-curve traits)	349
DiscrepancyStatistics (Statistic tool for sequences with discrepancy calculation)	350
DiscreteAveragingAsianOption (Discrete-averaging Asian option)	351
DiscreteAveragingAsianOption::arguments (Extra arguments for single-asset discrete-average Asian option)	353
DiscreteAveragingAsianOption::engine (Discrete-averaging Asian engine base class)	354
DiscreteGeometricASO (Discrete geometric average-strike Asian option (European style))	355
DiscretizedAsset (Discretized asset class used by numerical methods)	356
DiscretizedDiscountBond (Useful discretized discount bond asset)	359
DiscretizedOption (Discretized option on a given asset)	360
Disposable (Generic disposable object with move semantics)	362
DividendVanillaOption (Single-asset vanilla option (no barriers) with discrete dividends)	363
DividendVanillaOption::arguments (Arguments for dividend vanilla option calculation)	365
DividendVanillaOption::engine (Dividend vanilla option engine base class)	366
DKKCurrency (Danish krone)	367
DKKLibor (DKK LIBOR rate)	368
DMinus (D_- matricial representation)	369
DownRounding (Down-rounding)	370
DPlus (D_+ matricial representation)	371
DPlusDMinus (D_+D_- matricial representation)	372
DriftTermStructure (Drift term structure)	373
Duration (Duration type)	375
DZero (D_0 matricial representation)	376
EarlyExercise (Early-exercise base class)	377
EEKCurrency (Estonian kroon)	378
EndCriteria (Criteria to end optimization process)	379
EqualJumpsBinomialTree (Base class for equal jumps binomial tree)	381

EqualProbabilitiesBinomialTree (Base class for equal probabilities binomial tree)	382
Error (Base error class)	383
ErrorFunction (Error function)	384
ESPCurrency (Spanish peseta)	385
EulerDiscretization (Euler discretization for stochastic processes)	386
EURCurrency (European Euro)	388
Euribor (Euribor index)	389
EURLibor (EUR LIBOR rate)	390
EuropeanExercise (European exercise)	391
EuropeanOption (European option on a single asset)	392
ExchangeRate (Exchange rate between two currencies)	393
ExchangeRateManager (Exchange-rate repository)	395
Exercise (Base exercise class)	397
ExplicitEuler (Forward Euler scheme for finite difference methods)	398
ExtendedCoxIngersollRoss (Extended Cox-Ingersoll-Ross model class)	400
ExtendedCoxIngersollRoss::Dynamics (Short-rate dynamics in the extended Cox-Ingersoll-Ross model)	402
ExtendedCoxIngersollRoss::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	403
ExtendedDiscountCurve (Term structure based on loglinear interpolation of discount factors)	404
Extrapolator (Base class for classes possibly allowing extrapolation)	406
Factorial (Factorial numbers calculator)	407
FalsePosition (False position 1-D solver)	408
FaureRsg (Faure low-discrepancy sequence generator)	409
FDAmericanEngine (Finite-differences pricing engine for American one asset options) .	410
FDBermudanEngine (Finite-differences Bermudan engine)	411
FDDividendAmericanEngine (Finite-differences pricing engine for dividend American options)	412
FDDividendEngine (Base finite-differences pricing engine for dividend options)	413
FDDividendEuropeanEngine (Finite-differences pricing engine for dividend European options)	414
FDDividendShoutEngine (Finite-differences shout engine with dividends)	415
FDEuropeanEngine (Pricing engine for European options using finite-differences) . . .	416
FDShoutEngine (Finite-differences pricing engine for shout vanilla options)	417
FDStepConditionEngine (Finite-differences pricing engine for American-style vanilla options)	418
FDVanillaEngine (Finite-differences pricing engine for BSM one asset options)	419
FIMCurrency (Finnish markka)	421
FiniteDifferenceModel (Generic finite difference model)	422
FixedCouponBond (Fixed-coupon bond)	423
FixedCouponBondHelper (Fixed-coupon bond helper)	424
FixedRateCoupon (Coupon paying a fixed interest rate)	426
FlatForward (Flat interest-rate curve)	428
FloatingRateBond (Floating-rate bond)	430
FloatingRateCoupon (Coupon paying a variable rate)	431
Floor (Concrete floor class)	433
FloorTruncation (Floor truncation)	434
ForwardEngine (Forward engine base class)	435
ForwardFlat (Forward-flat interpolation factory and traits)	436
ForwardFlatInterpolation (Forward-flat interpolation between discrete points)	437
ForwardOptionArguments (Arguments for forward (strike-resetting) option calculation)	438
ForwardPerformanceEngine (Forward performance engine)	439
ForwardRate (Forward-curve traits)	440

ForwardRateStructure (Forward rate term structure)	441
ForwardSpreadedTermStructure (Term structure with added spread on the instantaneous forward rate)	443
ForwardVanillaOption (Forward version of a vanilla option)	445
FraRateHelper (Forward rate agreement)	447
FRFCurrency (French franc)	449
FuturesRateHelper (Interest-rate futures)	450
G2 (Two-additive-factor gaussian model class)	451
G2::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	453
G2SwaptionEngine (Swaption priced by means of the Black formula)	454
GammaFunction (Gamma function class)	455
GapPayoff (Binary gap payoff)	456
GaussChebyshev2thIntegration (Gauss-Chebyshev integration second kind)	457
GaussChebyshevIntegration (Gauss-Chebyshev integration)	458
GaussGegenbauerIntegration (Gauss-Gegenbauer integration)	459
GaussHermiteIntegration (Generalized Gauss-Hermite integration)	460
GaussHermitePolynomial (Gauss-Hermite polynomial)	461
GaussHyperbolicIntegration (Gauss-Hyperbolic integration)	462
GaussHyperbolicPolynomial (Gauss hyperbolic polynomial)	463
GaussianOrthogonalPolynomial (Orthogonal polynomial for Gaussian quadratures)	464
GaussianQuadrature (Integral of a 1-dimensional function using the Gauss quadratures method)	465
GaussianStatistics (Statistics tool for gaussian-assumption risk measures)	466
GaussJacobiIntegration (Gauss-Jacobi integration)	469
GaussJacobiPolynomial (Gauss-Jacobi polynomial)	470
GaussLaguerreIntegration (Generalized Gauss-Laguerre integration)	471
GaussLaguerrePolynomial (Gauss-Laguerre polynomial)	472
GaussLegendreIntegration (Gauss-Legendre integration)	473
GBPCurrency (British pound sterling)	474
GBPLibor (GBP LIBOR rate)	475
GeneralStatistics (Statistics tool)	476
GenericEngine (Template base class for option pricing engines)	479
GenericModelEngine (Base class for some pricing engine on a particular model)	480
GenericRiskStatistics (Empirical-distribution risk measures)	481
GeometricBrownianMotionProcess (Geometric brownian-motion process)	484
Germany (German calendars)	485
GRDCurrency (Greek drachma)	488
Greeks (Additional option results)	489
HaltonRsg (Halton low-discrepancy sequence generator)	490
Handle (Globally accessible relinkable pointer)	491
Helsinki (Helsinki calendar)	492
HestonModel (Heston model for the stochastic volatility of an asset)	493
HestonModelHelper (Calibration helper for Heston model)	494
HestonProcess (Square-root stochastic-volatility Heston process)	495
History (Container for historical data)	497
History::const_iterator (Random access iterator on history entries)	500
History::Entry (Single datum in history)	501
HKDCurrency (Honk Kong dollar)	502
HongKong (Hong Kong calendar)	503
HUFCurrency (Hungarian forint)	504
HullWhite (Single-factor Hull-White (extended Vasicek) model class)	505
HullWhite::Dynamics (Short-rate dynamics in the Hull-White model)	507
HullWhite::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	508
IEPCurrency (Irish punt)	509

ILSCurrency (Israeli shekel)	510
IMM (Main cycle of the International Money Market (a.k.a. IMM) Months)	511
ImplicitEuler (Backward Euler scheme for finite difference methods)	512
ImpliedTermStructure (Implied term structure at a given date in the future)	513
ImpliedVolTermStructure (Implied vol term structure at a given date in the future)	515
InArrearIndexedCoupon (In-arrear floating-rate coupon)	517
IncrementalStatistics (Statistics tool based on incremental accumulation)	519
Index (Purely virtual base class for indexes)	522
IndexedCoupon (Base indexed coupon class)	523
IndexManager (Global repository for past index fixings)	525
INRCurrency (Indian rupee)	526
Instrument (Abstract instrument class)	527
IntegralEngine	530
InterestRate (Concrete interest rate class)	531
InterpolatedDiscountCurve (Term structure based on interpolation of discount factors)	534
InterpolatedForwardCurve (Term structure based on interpolation of forward rates)	536
InterpolatedZeroCurve (Term structure based on interpolation of zero yields)	538
Interpolation (Base class for 1-D interpolations)	540
Interpolation2D (Base class for 2-D interpolations)	541
Interpolation2D::templateImpl (Basic template implementation)	542
Interpolation2DImpl (Abstract base class for 2-D interpolation implementations)	543
Interpolation::templateImpl (Basic template implementation)	544
InterpolationImpl (Abstract base class for interpolation implementations)	545
InverseCumulativeNormal (Inverse cumulative normal distribution function)	546
InverseCumulativePoisson (Inverse cumulative Poisson distribution function)	547
InverseCumulativeRng (Inverse cumulative random number generator)	548
InverseCumulativeRsg (Inverse cumulative random sequence generator)	549
IQDCurrency (Iraqi dinar)	550
IRRCurrency (Iranian rial)	551
ISKCurrency (Iceland krona)	552
Istanbul (Istanbul calendar)	553
Italy (Italian calendars)	554
ITLCurrency (Italian lira)	556
JamshidianSwaptionEngine (Jamshidian swaption engine)	557
JarrowRudd (Jarrow-Rudd (multiplicative) equal probabilities binomial tree)	558
Jibar (JIBAR rate)	559
Johannesburg (Johannesburg calendar)	560
JointCalendar (Joint calendar)	561
JPYCurrency (Japanese yen)	562
JPYLibor (JPY LIBOR rate)	563
JumpDiffusionEngine (Jump-diffusion engine for vanilla options)	564
JuQuadraticApproximationEngine	565
KnuthUniformRng (Uniform random number generator)	566
KronrodIntegral (Integral of a 1-dimensional function using the Gauss-Kronrod method)	567
KRWCurrency (South-Korean won)	568
KWDCurrency (Kuwaiti dinar)	569
Lattice (Lattice-method base class)	570
Lattice1D (One-dimensional lattice)	572
Lattice2D (Two-dimensional lattice)	573
LatticeShortRateModelEngine (Engine for a short-rate model specialized on a lattice)	574
LazyObject (Framework for calculation on demand and result caching)	575
LeastSquareFunction (Cost function for least-square problems)	577
LeastSquareProblem (Base class for least square problem)	578
LecuyerUniformRng (Uniform random number generator)	579

LeisenReimer (Leisen & Reimer tree: multiplicative approach)	580
LexicographicalView (Lexicographical 2-D view of a contiguous set of data)	581
Libor (Base class for BBA LIBOR indexes)	583
Linear (Linear interpolation factory and traits)	584
LinearInterpolation (Linear interpolation between discrete points)	585
LineSearch (Base class for line search)	586
Link (Relinkable access to a shared pointer)	588
LocalConstantVol (Constant local volatility, no time-strike dependence)	590
LocalVolCurve (Local volatility curve derived from a Black curve)	591
LocalVolSurface (Local volatility surface derived from a Black vol surface)	593
LocalVolTermStructure (Local-volatility term structure)	595
LogLinear (Log-linear interpolation factory and traits)	597
LogLinearInterpolation	598
LTLCurrency (Lithuanian litas)	599
LUFCurrency (Luxembourg franc)	600
LVLCurrency (Latvian lat)	601
MakeMCDigitalEngine (Monte Carlo digital engine factory)	602
MakeMCEuropeanEngine (Monte Carlo European engine factory)	603
MakeMCEuropeanHestonEngine (Monte Carlo Heston European engine factory)	604
MakeSchedule (Helper class)	605
Matrix (Matrix used in linear algebra)	606
MCAmericanBasketEngine (Least-square Monte Carlo engine)	610
MCBarrierEngine (Pricing engine for barrier options using Monte Carlo simulation)	611
MCBasketEngine (Pricing engine for basket options using Monte Carlo simulation)	613
McCliquetOption (Simple example of Monte Carlo pricer)	615
MCDigitalEngine (Pricing engine for digital options using Monte Carlo simulation)	616
MCDiscreteArithmeticAPEngine (Monte Carlo pricing engine for discrete arithmetic average price Asian)	617
McDiscreteArithmeticASO (Example of Monte Carlo pricer using a control variate)	619
MCDiscreteAveragingAsianEngine (Pricing engine for discrete average Asians using Monte Carlo simulation)	620
MCDiscreteGeometricAPEngine (Monte Carlo pricing engine for discrete geometric average price Asian)	622
MCEuropeanEngine (European option pricing engine using Monte Carlo simulation)	623
MCEuropeanHestonEngine (Monte Carlo Heston-model engine for European options)	624
McEverest (Everest-type option pricer)	625
MCHestonEngine (Monte Carlo Heston-model engine)	626
McHimalaya (Himalayan-type option pricer)	627
McMaxBasket (Simple example of multi-factor Monte Carlo pricer)	628
McPagoda (Roofed Asian option)	629
McPerformanceOption (Performance option computed using Monte Carlo simulation)	630
McPricer (Base class for Monte Carlo pricers)	631
McSimulation (Base class for Monte Carlo engines)	632
MCVanillaEngine (Pricing engine for vanilla options using Monte Carlo simulation)	634
MersenneTwisterUniformRng (Uniform random number generator)	636
Merton76Process (Merton-76 jump-diffusion process)	637
MixedScheme (Mixed (explicit/implicit) scheme for finite difference methods)	639
Money (Amount of cash)	641
MonotonicCubicSpline (Cubic spline with monotonicity constraint)	643
MonteCarloModel (General purpose Monte Carlo model for path samples)	644
MoreGreeks (More additional option results)	645
MoroInverseCumulativeNormal (Moro Inverse cumulative normal distribution class)	646
MTLCurrency (Maltese lira)	647
MultiAsset	648

MultiAssetOption (Base class for options on multiple assets)	649
MultiAssetOption::arguments (Arguments for multi-asset option calculation)	651
MultiAssetOption::results (Results from multi-asset option calculation)	652
MultiCubicSpline	653
MultiPath (Correlated multiple asset paths)	654
MultiPathGenerator (Generates a multipath from a random number generator)	655
MultiVariate (Default Monte Carlo traits for multi-variate models)	656
MXNCurrency (Mexican peso)	657
NaturalCubicSpline (Cubic spline with null second derivative at end points)	658
NaturalMonotonicCubicSpline (Natural cubic spline with monotonicity constraint)	659
NeumannBC (Neumann boundary condition (i.e., constant derivative))	660
Newton (Newton 1-D solver)	662
NewtonSafe (Safe Newton 1-D solver)	663
NLGCurrency (Dutch guilder)	664
NoConstraint (No constraint)	665
NOKCurrency (Norwegian krone)	666
NonLinearLeastSquare (Non-linear least-square method)	667
NormalDistribution (Normal distribution function)	668
NPRCurrency (Nepal rupee)	669
Null (Template class providing a null value for a given type)	670
NullCalendar (Calendar for reproducing theoretical calculations)	671
NullCondition (Null step condition)	672
NullParameter (Parameter which is always zero $a(t) = 0$)	673
NumericalMethod (Numerical method (tree, finite-differences) base class)	674
NZDCurrency (New Zealand dollar)	676
NZDLibor (NZD LIBOR rate)	677
Observable (Object that notifies its changes to a set of observables)	678
ObservableValue (Observable and assignable proxy to concrete value)	679
Observer (Object that gets notified when a given observable changes)	680
OneAssetOption (Base class for options on a single asset)	682
OneAssetOption::arguments (Arguments for single-asset option calculation)	685
OneAssetOption::results (Results from single-asset option calculation)	686
OneAssetStrikedOption (Base class for options on a single asset with striked payoff)	687
OneDayCounter (1/1 day count convention)	689
OneFactorAffineModel (Single-factor affine base class)	690
OneFactorModel (Single-factor short-rate model abstract class)	691
OneFactorModel::ShortRateDynamics (Base class describing the short-rate dynamics)	692
OneFactorModel::ShortRateTree (Recombining trinomial tree discretizing the state variable)	693
OneFactorOperator (Interest-rate single factor model differential operator)	694
OptimizationMethod (Abstract class for constrained optimization method)	695
Option (Base option class)	697
Option::arguments	698
OrnsteinUhlenbeckProcess (Ornstein-Uhlenbeck process class)	699
Oslo (Oslo calendar)	701
Parameter (Base class for model arguments)	702
ParameterImpl (Base class for model parameter implementation)	703
ParCoupon (coupon at par on a term structure)	704
Path	706
PathGenerator (Generates random paths using a sequence generator)	707
PathPricer (Base class for path pricers)	708
Payoff (Base class for option payoffs)	709
PercentageStrikePayoff (Payoff with strike expressed as percentage)	710
Period (Time period described by a number of a given time unit)	711

PiecewiseConstantParameter (Piecewise-constant parameter)	712
PiecewiseYieldCurve (Piecewise yield term structure)	713
PKRCurrency (Pakistani rupee)	715
PlainVanillaPayoff (Plain-vanilla payoff)	716
PLNCurrency (Polish zloty)	717
PoissonDistribution (Normal distribution function)	718
PositiveConstraint (Constraint imposing positivity to all arguments)	719
Prague (Prague calendar)	720
PricingEngine (Interface for pricing engines)	721
PrimeNumbers (Prime numbers calculator)	722
Problem (Constrained optimization problem)	723
PTECurrency (Portuguese escudo)	724
QuantoEngine (Quanto engine base class)	725
QuantoForwardVanillaOption (Quanto version of a forward vanilla option)	727
QuantoOptionArguments (Arguments for quanto option calculation)	729
QuantoOptionResults (Results from quanto option calculation)	730
QuantoTermStructure (Quanto term structure)	731
QuantoVanillaOption (Quanto version of a vanilla option)	733
Quote (Purely virtual base class for market observables)	735
RandomizedLDS (Randomized (random shift) low-discrepancy sequence)	736
RandomSequenceGenerator (Random sequence generator based on a pseudo-random number generator)	738
RateHelper (Base class for rate helpers)	739
Results (Base class for generic result groups)	741
Ridder (Ridder 1-D solver)	742
Riyadh (Riyadh calendar)	743
ROLCurrency (Romanian leu)	744
Rounding (Basic rounding class)	745
SalvagingAlgorithm (Algorithm used for matricial pseudo square root)	747
Sample (Weighted sample)	748
SampledCurve (This class contains a sampled curve)	749
SARCCurrency (Saudi riyal)	750
Schedule (Payment schedule)	751
Secant (Secant 1-D solver)	752
SeedGenerator (Random seed generator)	753
SegmentIntegral (Integral of a one-dimensional function)	754
SEKCurrency (Swedish krona)	755
Seoul (Seoul calendar)	756
SequenceStatistics (Statistics analysis of N-dimensional (sequence) data)	757
Settings (Global repository for run-time library settings)	759
SGDCurrency (Singapore dollar)	761
Short (Short indexed coupon)	762
Short< ParCoupon > (Short coupon at par on a term structure)	763
ShortRateModel (Abstract short-rate model class)	764
ShoutCondition (Shout option condition)	766
SimpleCashFlow (Predetermined cash flow)	767
SimpleDayCounter (Simple day counter for reproducing theoretical calculations)	768
SimpleQuote (Market element returning a stored value)	769
SimpleSwap (Simple fixed-rate vs Libor swap)	770
SimpleSwap::arguments (Arguments for simple swap calculation)	772
SimpleSwap::results (Results from simple swap calculation)	773
Simplex (Multi-dimensional simplex class)	774
SimpsonIntegral (Integral of a one-dimensional function)	775
Singapore (Singapore calendar)	776

SingleAsset	777
SingleAssetOption (Black-Scholes-Merton option)	778
Singleton (Basic support for the singleton pattern)	780
SingleVariate (Default Monte Carlo traits for single-variate models)	781
SITCurrency (Slovenian tolar)	782
SKKCurrency (Slovak koruna)	783
SobolRsg (Sobol low-discrepancy sequence generator)	784
Solver1D (Base class for 1-D solvers)	786
SquareRootProcess (Square-root process class)	788
StatsHolder (Helper class for precomputed distributions)	789
SteepestDescent (Multi-dimensional steepest-descent class)	790
step_iterator (Iterator advancing in constant steps)	791
StepCondition (Condition to be applied at every time step)	792
StepConditionSet (Parallel evolver for multiple arrays)	793
StochasticProcess (Multi-dimensional stochastic process class)	794
StochasticProcess1D (1-dimensional stochastic process)	797
StochasticProcess1D::discretization (Discretization of a 1-D stochastic process)	799
StochasticProcess::discretization (Discretization of a stochastic process over a given time interval)	800
StochasticProcessArray (Array of correlated 1-D stochastic processes)	801
Stock (Simple stock class)	803
Stockholm (Stockholm calendar)	804
StrikedTypePayoff (Intermediate class for payoffs based on a fixed strike)	805
StulzEngine (Pricing engine for 2D European Baskets)	806
SuperSharePayoff (Binary supershare payoff)	807
SVD (Singular value decomposition)	808
Swap (Interest rate swap)	809
SwapRateHelper (Swap rate)	811
Swaption (Swaption class)	813
Swaption::arguments (Arguments for swaption calculation)	815
Swaption::results (Results from swaption calculation)	816
SwaptionVolatilityMatrix (At-the-money swaption-volatility matrix)	817
SwaptionVolatilityStructure (Swaption-volatility structure)	819
Sydney (Sydney calendar (New South Wales, Australia))	821
SymmetricSchurDecomposition (Symmetric threshold Jacobi algorithm)	822
TabulatedGaussLegendre (Tabulated Gauss-Legendre quadratures)	823
Taipei (Taipei calendar)	824
Taiwan (Taiwan calendar)	825
TARGET (TARGET calendar)	826
TermStructure (Basic term-structure functionality)	827
TermStructureConsistentModel (Term-structure consistent model class)	829
TermStructureFittingParameter (Deterministic time-dependent parameter used for yield-curve fitting)	830
THBCurrency (Thai baht)	831
Thirty360 (30/360 day count convention)	832
Tian (Tian tree: third moment matching, multiplicative approach)	833
Tibor (JPY TIBOR index)	834
TimeBasket (Distribution over a number of dates)	835
TimeGrid (Time grid class)	836
Tokyo (Tokyo calendar)	838
Toronto (Toronto calendar)	840
TqrEigenDecomposition (Tridiag. QR eigen decomposition with explicite shift aka Wilkinson)	841
TrapezoidIntegral (Integral of a one-dimensional function)	842

Tree (Tree approximating a single-factor diffusion)	844
TreeCapFloorEngine (Numerical lattice engine for cap/floors)	845
TreeSwaptionEngine (Numerical lattice engine for swaptions)	846
TridiagonalOperator (Base implementation for tridiagonal operator)	847
TridiagonalOperator::TimeSetter (Encapsulation of time-setting logic)	849
Trigeorgis (Trigeorgis (additive equal jumps) binomial tree)	850
TrinomialTree (Recombining trinomial tree class)	851
TRLCurrency (Turkish lira)	852
TRLibor (TRY LIBOR rate)	853
TRYCurrency (New Turkish lira)	854
TTDCurrency (Trinidad & Tobago dollar)	855
TWDCurrency (Taiwan dollar)	856
TwoFactorModel (Abstract base-class for two-factor models)	857
TwoFactorModel::ShortRateDynamics (Class describing the dynamics of the two state variables)	858
TwoFactorModel::ShortRateTree (Recombining two-dimensional tree discretizing the state variable)	859
TypePayoff (Intermediate class for call/put/straddle payoffs)	860
UnitedKingdom (United Kingdom calendars)	861
UnitedStates (United States calendars)	863
UpFrontIndexedCoupon (up front indexed coupon class)	866
UpRounding (Up-rounding)	867
USDCurrency (U.S. dollar)	868
USDLibor (USD LIBOR rate)	869
Value (Pricing results)	870
VanillaOption (Vanilla option (no discrete dividends, no barriers) on a single asset)	871
VanillaOption::engine (Vanilla option engine base class)	872
Vasicek (Vasicek model class)	873
Vasicek::Dynamics (Short-rate dynamics in the Vasicek model)	875
VEBCurrency (Venezuelan bolivar)	876
Visitor (Visitor for a specific class)	877
Warsaw (Warsaw calendar)	878
Wellington (Wellington calendar)	879
Xibor (Base class for LIBOR-like indexes)	880
YieldTermStructure (Interest-rate term structure)	882
ZARCurrency (South-African rand)	886
ZeroCouponBond (Zero-coupon bond)	887
ZeroSpreadedTermStructure (Term structure with an added spread on the zero yield rate)	888
ZeroYield (Zero-curve traits)	890
ZeroYieldStructure (Zero-yield term structure)	891
Zibor (CHF ZIBOR rate)	893
Zurich (Zurich calendar)	894

Chapter 5

QuantLib File Index

5.1 QuantLib File List

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

ql/argsandresults.hpp (Base classes for generic arguments and results)	895
ql/calendar.hpp (calendar class)	897
ql/capvolstructures.hpp (Cap/Floor volatility structures)	928
ql/cashflow.hpp (Base class for cash flows)	929
ql/currency.hpp (Known currencies)	955
ql/date.hpp (Date- and time-related classes, typedefs and enumerations)	956
ql/daycounter.hpp (Day counter class)	959
ql/discretizedasset.hpp (Discretized asset classes)	966
ql/errors.hpp (Classes and functions for error handling)	967
ql/exchangerate.hpp (Exchange rate between two currencies)	970
ql/exercise.hpp (Option exercise classes and payoff function)	971
ql/grid.hpp (Grid constructors)	993
ql/handle.hpp (Globally accessible relinkable pointer)	994
ql/history.hpp (History class)	995
ql/index.hpp (Purely virtual base class for indexes)	996
ql/instrument.hpp (Abstract instrument class)	1015
ql/interestrates.hpp (Instrument rate class)	1042
ql/money.hpp (Cash amount in a given currency)	1102
ql/numericalmethod.hpp (Numerical method class)	1114
ql/option.hpp (Base option class)	1126
ql/payoff.hpp (Option payoff classes)	1134
ql/pricingengine.hpp (Base class for pricing engines)	1145
ql/qldefines.hpp (Global definitions and compiler switches)	1217
ql/quote.hpp (Purely virtual base class for market observables)	1219
ql/schedule.hpp (Date schedule)	1235
ql/settings.hpp (Global repository for run-time library settings)	1236
ql/solver1d.hpp (Abstract 1-D solver class)	1253
ql/stochasticprocess.hpp (Stochastic processes)	1261
ql/swaptionvolstructure.hpp (Swaption volatility structure)	1262
ql/termstructure.hpp (Base class for term structures)	1263
ql/timegrid.hpp (Discrete time grid)	1283
ql/types.hpp (Custom types)	1284
ql/voltermstructure.hpp (Volatility term structures)	1306

ql/ yieldtermstructure.hpp (Interest-rate term structure)	1307
ql/Calendars/ beijing.hpp (Beijing calendar)	898
ql/Calendars/ bombay.hpp (Bombay calendar)	899
ql/Calendars/ bratislava.hpp (Bratislava calendar)	900
ql/Calendars/ budapest.hpp (Budapest calendar)	901
ql/Calendars/ copenhagen.hpp (Copenhagen calendar)	902
ql/Calendars/ germany.hpp (German calendars)	903
ql/Calendars/ helsinki.hpp (Helsinki calendar)	904
ql/Calendars/ hongkong.hpp (Hong Kong calendar)	905
ql/Calendars/ istanbul.hpp (Istanbul calendar)	906
ql/Calendars/ italy.hpp (Italian calendars)	907
ql/Calendars/ johannesburg.hpp (Johannesburg calendar)	908
ql/Calendars/ jointcalendar.hpp (Joint calendar)	909
ql/Calendars/ nullcalendar.hpp (Calendar for reproducing theoretical calculations)	910
ql/Calendars/ oslo.hpp (Oslo calendar)	911
ql/Calendars/ prague.hpp (Prague calendar)	912
ql/Calendars/ riyadh.hpp (Riyadh calendar)	913
ql/Calendars/ seoul.hpp (South Korea calendar)	914
ql/Calendars/ singapore.hpp (Singapore calendar)	915
ql/Calendars/ stockholm.hpp (Stockholm calendar)	916
ql/Calendars/ sydney.hpp (Sydney calendar)	917
ql/Calendars/ taipei.hpp (Taipei calendar)	918
ql/Calendars/ taiwan.hpp (Taiwan calendar)	919
ql/Calendars/ target.hpp (TARGET calendar)	920
ql/Calendars/ tokyo.hpp (Tokyo calendar)	921
ql/Calendars/ toronto.hpp (Toronto calendar)	922
ql/Calendars/ unitedkingdom.hpp (UK calendars)	923
ql/Calendars/ unitedstates.hpp (US calendars)	924
ql/Calendars/ warsaw.hpp (Warsaw calendar)	925
ql/Calendars/ wellington.hpp (Wellington calendar)	926
ql/Calendars/ zurich.hpp (Zurich calendar)	927
ql/CashFlows/ analysis.hpp (Cash-flow analysis functions)	930
ql/CashFlows/ basispointsensitivity.hpp (Basis point sensitivity calculator)	931
ql/CashFlows/ cashflowvectors.hpp (Cash flow vector builders)	932
ql/CashFlows/ coupon.hpp (Coupon accruing over a fixed period)	933
ql/CashFlows/ fixedratecoupon.hpp (Coupon paying a fixed annual rate)	934
ql/CashFlows/ floatingratecoupon.hpp (Coupon paying a variable rate)	935
ql/CashFlows/ inrearindexedcoupon.hpp (In-arrear floating-rate coupon)	936
ql/CashFlows/ indexedcashflowvectors.hpp (Indexed cash-flow vector builders)	937
ql/CashFlows/ indexedcoupon.hpp (Indexed coupon)	938
ql/CashFlows/ parcoupon.hpp (Coupon at par on a term structure)	939
ql/CashFlows/ shortfloatingcoupon.hpp (Short (or long) coupon at par on a term structure)	940
ql/CashFlows/ shortindexedcoupon.hpp (Short (or long) indexed coupon)	941
ql/CashFlows/ simplecashflow.hpp (Predetermined cash flow)	942
ql/CashFlows/ timebasket.hpp	943
ql/CashFlows/ upfrontindexedcoupon.hpp (Up front indexed coupon)	944
ql/Currencies/ africa.hpp (African currencies)	945
ql/Currencies/ america.hpp (American currencies)	946
ql/Currencies/ asia.hpp (Asian currencies)	948
ql/Currencies/ europe.hpp (European currencies)	950
ql/Currencies/ exchangeratemanager.hpp (Exchange-rate repository)	953
ql/Currencies/ oceania.hpp (Oceanian currencies)	954
ql/DayCounters/ actual360.hpp (Act/360 day counter)	960
ql/DayCounters/ actual365fixed.hpp (Actual/365 (Fixed) day counter)	961

ql/DayCounters/actualactual.hpp (Act/act day counters)	962
ql/DayCounters/one.hpp (1/1 day counter)	963
ql/DayCounters/simpliedaycounter.hpp (Simple day counter for reproducing theoretical calculations)	964
ql/DayCounters/thirty360.hpp (30/360 day counters)	965
ql/FiniteDifferences/americancondition.hpp (American option exercise condition)	972
ql/FiniteDifferences/boundarycondition.hpp (Boundary conditions for differential operators)	973
ql/FiniteDifferences/bsmoperator.hpp (Differential operator for Black-Scholes-Merton equation)	974
ql/FiniteDifferences/bsmtermoperator.hpp (Differential operator for Black-Scholes-Merton equation)	975
ql/FiniteDifferences/cranknicolson.hpp (Crank-Nicolson scheme for finite difference methods)	976
ql/FiniteDifferences/dminus.hpp (D_- matricial representation)	977
ql/FiniteDifferences/dplus.hpp (D_+ matricial representation)	978
ql/FiniteDifferences/dplusdminus.hpp (D_+D_- matricial representation)	979
ql/FiniteDifferences/dzero.hpp (D_0 matricial representation)	980
ql/FiniteDifferences/expliciteuler.hpp (Explicit Euler scheme for finite difference methods)	981
ql/FiniteDifferences/fdtypedefs.hpp (Default choices for template instantiations)	982
ql/FiniteDifferences/finitedifferencemodel.hpp (Generic finite difference model)	983
ql/FiniteDifferences/impliciteuler.hpp (Implicit Euler scheme for finite difference methods)	984
ql/FiniteDifferences/mixedscheme.hpp (Mixed (explicit/implicit) scheme for finite difference methods)	985
ql/FiniteDifferences/onefactoroperator.hpp (General differential operator for one-factor interest rate models)	986
ql/FiniteDifferences/operatortraits.hpp (Differential operator traits)	987
ql/FiniteDifferences/parallelevolver.hpp (Parallel evolver for multiple arrays)	988
ql/FiniteDifferences/shoutcondition.hpp (Shout option exercise condition)	989
ql/FiniteDifferences/stepcondition.hpp (Conditions to be applied at every time step)	990
ql/FiniteDifferences/tridiagonaloperator.hpp (Tridiagonal operator)	991
ql/FiniteDifferences/valueatcenter.hpp (Compute value, first, and second derivatives at grid center)	992
ql/Indexes/audlibor.hpp (AUD LIBOR rate)	997
ql/Indexes/cadlibor.hpp (CAD LIBOR rate)	998
ql/Indexes/cdor.hpp (CDOR rate)	999
ql/Indexes/chflibor.hpp (CHF LIBOR rate)	1000
ql/Indexes/dkklbtor.hpp (DKK LIBOR rate)	1001
ql/Indexes/euribor.hpp (Euribor index)	1002
ql/Indexes/eurlibor.hpp (EUR LIBOR rate)	1003
ql/Indexes/gbplibor.hpp (GBP LIBOR rate)	1004
ql/Indexes/indexmanager.hpp (Global repository for past index fixings)	1005
ql/Indexes/jibar.hpp (JIBAR rate)	1006
ql/Indexes/jpylibor.hpp (JPY LIBOR rate)	1007
ql/Indexes/libor.hpp (Base class for BBA LIBOR indexes)	1008
ql/Indexes/nzdlbtor.hpp (NZD LIBOR rate)	1009
ql/Indexes/tibor.hpp (JPY TIBOR rate)	1010
ql/Indexes/trlibor.hpp (TRY LIBOR rate)	1011
ql/Indexes/usdlbtor.hpp (USD LIBOR rate)	1012
ql/Indexes/xibor.hpp (Base class for LIBOR-like indexes)	1013
ql/Indexes/zibor.hpp (CHF ZIBOR rate)	1014
ql/Instruments/asianoption.hpp (Asian option on a single asset)	1016

ql/Instruments/ barrieroption.hpp (Barrier option on a single asset)	1017
ql/Instruments/ basketoption.hpp (Basket option on a number of assets)	1018
ql/Instruments/ bond.hpp (Concrete bond class)	1019
ql/Instruments/ callabilityschedule.hpp (Schedule of put/call dates)	1020
ql/Instruments/ capfloor.hpp (Cap and Floor class)	1021
ql/Instruments/ cliquetoption.hpp (Cliquet option)	1022
ql/Instruments/ convertiblebond.hpp (Convertible bond)	1023
ql/Instruments/ dividendschedule.hpp (Schedule of dividend dates)	1024
ql/Instruments/ dividendvanillaoption.hpp (Vanilla option on a single asset with discrete dividends)	1025
ql/Instruments/ europeanoption.hpp (European option on a single asset)	1026
ql/Instruments/ fixedcouponbond.hpp (Fixed-coupon bond)	1027
ql/Instruments/ floatingratebond.hpp (Floating-rate bond)	1028
ql/Instruments/ forwardvanillaoption.hpp (Forward version of a vanilla option)	1029
ql/Instruments/ multiassetoption.hpp (Option on multiple assets)	1030
ql/Instruments/ oneassetoption.hpp (Option on a single asset)	1031
ql/Instruments/ oneassetstrikedoption.hpp (Option on a single asset with striked payoff)	1032
ql/Instruments/ payoffs.hpp (Payoffs for various options)	1033
ql/Instruments/ quantoforwardvanillaoption.hpp (Quanto version of a forward vanilla option)	1034
ql/Instruments/ quantovanillaoption.hpp (Quanto version of a vanilla option)	1035
ql/Instruments/ simpleswap.hpp (Simple fixed-rate vs Libor swap)	1036
ql/Instruments/ stock.hpp (Concrete stock class)	1037
ql/Instruments/ swap.hpp (Interest rate swap)	1038
ql/Instruments/ swaption.hpp (Swaption class)	1039
ql/Instruments/ vanillaoption.hpp (Vanilla option on a single asset)	1040
ql/Instruments/ zerocouponbond.hpp (Zero-coupon bond)	1041
ql/Lattices/ binomialtree.hpp (Binomial tree class)	1043
ql/Lattices/ bsmlattice.hpp (Binomial trees under the BSM model)	1044
ql/Lattices/ lattice.hpp (Lattice method class)	1045
ql/Lattices/ lattice1d.hpp (One-dimensional lattice class)	1046
ql/Lattices/ lattice2d.hpp (Two-dimensional lattice class)	1047
ql/Lattices/ tree.hpp (Tree class)	1048
ql/Lattices/ trinomialtree.hpp (Trinomial tree class)	1049
ql/Math/ array.hpp (1-D array used in linear algebra)	1050
ql/Math/ backwardflatinterpolation.hpp (Backward-flat interpolation between discrete points)	1051
ql/Math/ beta.hpp (Beta and beta incomplete functions)	1052
ql/Math/ bicubicsplineinterpolation.hpp (Bicubic spline interpolation between discrete points)	1053
ql/Math/ bilinearinterpolation.hpp (Bilinear interpolation between discrete points) . . .	1054
ql/Math/ binomialdistribution.hpp (Binomial distribution)	1055
ql/Math/ bivariatenormaldistribution.hpp (Bivariate cumulative normal distribution) . .	1056
ql/Math/ chisquaredistribution.hpp (Chi-square (central and non-central) distributions)	1057
ql/Math/ choleskydecomposition.hpp (Cholesky decomposition)	1058
ql/Math/ comparison.hpp (Floating-point comparisons)	1059
ql/Math/ convergencestatistics.hpp (Statistics tool with risk measures)	1060
ql/Math/ cubicspline.hpp (Cubic spline interpolation between discrete points)	1061
ql/Math/ discrepancystatistics.hpp (Statistic tool for sequences with discrepancy calcula- tion)	1062
ql/Math/ errorfunction.hpp (Error function)	1063
ql/Math/ extrapolation.hpp (Class-wide extrapolation settings)	1064
ql/Math/ factorial.hpp (Factorial numbers calculator)	1065

ql/Math/ forwardflatinterpolation.hpp (Forward-flat interpolation between discrete points)	1066
ql/Math/ functional.hpp (Functionals and combinators not included in the STL)	1067
ql/Math/ gammadistribution.hpp (Gamma distribution)	1068
ql/Math/ gaussianorthogonalpolynomial.hpp (Orthogonal polynomials for gaussian quadratures)	1069
ql/Math/ gaussianquadratures.hpp (Integral of a 1-dimensional function using the Gauss quadratures)	1070
ql/Math/ gaussianstatistics.hpp (Statistics tool for gaussian-assumption risk measures) .	1072
ql/Math/ generalstatistics.hpp (Statistics tool)	1073
ql/Math/ incompletegamma.hpp (Incomplete Gamma function)	1074
ql/Math/ incrementalstatistics.hpp (Statistics tool based on incremental accumulation) .	1075
ql/Math/ interpolation.hpp (Base class for 1-D interpolations)	1076
ql/Math/ interpolation2D.hpp (Abstract base classes for 2-D interpolations)	1077
ql/Math/ kronrodintegral.hpp (Integral of a 1-dimensional function using the Gauss-Kronrod method)	1078
ql/Math/ lexicographicalview.hpp (Lexicographical 2-D view of a contiguous set of data)	1079
ql/Math/ linearinterpolation.hpp (Linear interpolation between discrete points)	1080
ql/Math/ loglinearinterpolation.hpp (Log-linear interpolation between discrete points) .	1081
ql/Math/ matrix.hpp (Matrix used in linear algebra)	1082
ql/Math/ multicubicspline.hpp (N-dimensional cubic spline interpolation between discrete points)	1083
ql/Math/ normaldistribution.hpp (Normal, cumulative and inverse cumulative distributions)	1085
ql/Math/ poissondistribution.hpp (Poisson distribution)	1086
ql/Math/ primenumbers.hpp (Prime numbers calculator)	1087
ql/Math/ pseudosqrt.hpp (Pseudo square root of a real symmetric matrix)	1088
ql/Math/ riskstatistics.hpp (Empirical-distribution risk measures)	1089
ql/Math/ rounding.hpp (Rounding implementation)	1090
ql/Math/ sampledcurve.hpp (Class that contains a sampled curve)	1091
ql/Math/ segmentintegral.hpp (Integral of a one-dimensional function)	1092
ql/Math/ sequencestatistics.hpp (Statistics tools for sequence (vector, list, array) samples)	1093
ql/Math/ simpsonintegral.hpp (Integral of a one-dimensional function)	1095
ql/Math/ statistics.hpp (Statistics tool with risk measures)	1096
ql/Math/ svd.hpp (Singular value decomposition)	1097
ql/Math/ symmetriceigenvalues.hpp (Eigenvalues / eigenvectors of a real symmetric matrix)	1098
ql/Math/ symmetricschurdecomposition.hpp (Eigenvalues / eigenvectors of a real symmetric matrix)	1099
ql/Math/ tqreigendecomposition.hpp (Tridiag. QR eigen decomposition with explicit shift aka Wilkinson)	1100
ql/Math/ trapezoidintegral.hpp (Integral of a one-dimensional function)	1101
ql/MonteCarlo/ brownianbridge.hpp (Browian bridge)	1103
ql/MonteCarlo/ getcovariance.hpp (Covariance matrix calculation)	1104
ql/MonteCarlo/ mctraits.hpp (Monte Carlo policies)	1105
ql/MonteCarlo/ mctypedefs.hpp (Default choices for template instantiations)	1106
ql/MonteCarlo/ montecarlomodel.hpp (General purpose Monte Carlo model)	1107
ql/MonteCarlo/ multipath.hpp (Correlated multiple asset paths)	1108
ql/MonteCarlo/ multipathgenerator.hpp (Generates a multi path from a random-array generator)	1109
ql/MonteCarlo/ path.hpp (Single factor random walk)	1110
ql/MonteCarlo/ pathgenerator.hpp (Generates random paths using a sequence generator)	1111
ql/MonteCarlo/ pathpricer.hpp (Base class for single-path pricers)	1112
ql/MonteCarlo/ sample.hpp (Weighted sample)	1113

ql/Optimization/armijo.hpp (Armijo line-search class)	1115
ql/Optimization/conjugategradient.hpp (Conjugate gradient optimization method)	1116
ql/Optimization/constraint.hpp (Abstract constraint class)	1117
ql/Optimization/costfunction.hpp (Optimization cost function class)	1118
ql/Optimization/criteria.hpp (Optimization criteria class)	1119
ql/Optimization/leastsquare.hpp (Least square cost function)	1120
ql/Optimization/linesearch.hpp (Line search abstract class)	1121
ql/Optimization/method.hpp (Abstract optimization method class)	1122
ql/Optimization/problem.hpp (Abstract optimization class)	1123
ql/Optimization/simplex.hpp (Simplex optimization method)	1124
ql/Optimization/steepestdescent.hpp (Steepest descent optimization method)	1125
ql/Patterns/bridge.hpp (Bridge pattern (a.k.a. handle-body idiom))	1127
ql/Patterns/composite.hpp (Composite pattern)	1128
ql/Patterns/curiouslyrecurring.hpp (Curiously recurring template pattern)	1129
ql/Patterns/lazyobject.hpp (Framework for calculation on demand and result caching)	1130
ql/Patterns/observable.hpp (Observer/observable pattern)	1131
ql/Patterns/singleton.hpp (Basic support for the singleton pattern)	1132
ql/Patterns/visitor.hpp (Degenerate base class for the Acyclic Visitor pattern)	1133
ql/Pricers/discretegeometricaso.hpp (Discrete Geometric Average Strike Option)	1135
ql/Pricers/mccliqetoption.hpp (Cliquet option priced with Monte Carlo simulation)	1136
ql/Pricers/mcdiscretearithmeticaso.hpp (Discrete Arithmetic Average Strike Option)	1137
ql/Pricers/mceverest.hpp (Everest-type option pricer)	1138
ql/Pricers/mchimalaya.hpp (Himalayan-type option pricer)	1139
ql/Pricers/mcmaxbasket.hpp (Max Basket Monte Carlo pricer)	1140
ql/Pricers/mcpagoda.hpp (Roofed multi asset Asian option)	1141
ql/Pricers/mcperformanceoption.hpp (Performance option priced with Monte Carlo simulation)	1142
ql/Pricers/mcpricer.hpp (Base class for Monte Carlo pricers)	1143
ql/Pricers/singleassetoption.hpp (Common code for option evaluation)	1144
ql/PricingEngines/americanpayoffatexpiry.hpp (Analytical formulae for american exercise with payoff at expiry)	1146
ql/PricingEngines/americanpayoffathit.hpp (Analytical formulae for american exercise with payoff at hit)	1147
ql/PricingEngines/blackformula.hpp (Black formula)	1158
ql/PricingEngines/blackmodel.hpp (Abstract class for Black-type models (market models))	1159
ql/PricingEngines/genericmodelengine.hpp (Generic option engine based on a model)	1169
ql/PricingEngines/greeks.hpp (Default greek calculations)	1170
ql/PricingEngines/latticeshortratemodelengine.hpp (Engine for a short-rate model specialized on a lattice)	1171
ql/PricingEngines/mcsimulation.hpp (Framework for Monte Carlo engines)	1172
ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp (Analytic engine for continuous geometric average price Asian)	1148
ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp (Analytic engine for discrete geometric average price Asian)	1149
ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp (Monte Carlo engine for discrete arithmetic average price Asian)	1150
ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp (Monte Carlo engine for discrete geometric average price Asian)	1151
ql/PricingEngines/Asian/mcdiscreteasianengine.hpp (Monte Carlo pricing engine for discrete average Asians)	1152
ql/PricingEngines/Barrier/analyticbarrierengine.hpp (Analytic barrier option engines)	1153
ql/PricingEngines/Barrier/mcbarrierengine.hpp (Monte Carlo barrier option engines)	1154

ql/PricingEngines/Basket/ mcamericanbasketengine.hpp (Least-square Monte Carlo engines)	1155
ql/PricingEngines/Basket/ mcbasketengine.hpp (European basket MC Engine)	1156
ql/PricingEngines/Basket/ stulzengine.hpp (2D European Basket formulae, due to Stulz (1982))	1157
ql/PricingEngines/CapFloor/ analyticcapfloorengine.hpp (Analytic engine for caps/floors)	1160
ql/PricingEngines/CapFloor/ blackcapfloorengine.hpp (Black-formula cap/floor engine)	1161
ql/PricingEngines/CapFloor/ discretizedcapfloor.hpp (Discretized cap/floor)	1162
ql/PricingEngines/CapFloor/ treecapfloorengine.hpp (Numerical lattice engine for cap/floors)	1163
ql/PricingEngines/Cliquet/ analyticcliquetengine.hpp (Analytic Cliquet engine)	1164
ql/PricingEngines/Cliquet/ analyticperformanceengine.hpp (Analytic performance engine)	1165
ql/PricingEngines/Cliquet/ mccliquetengine.hpp (Monte Carlo Cliquet option engine)	1166
ql/PricingEngines/Forward/ forwardengine.hpp (Forward (strike-resetting) option engine)	1167
ql/PricingEngines/Forward/ forwardperformanceengine.hpp (Forward (strike-resetting) performance option engines)	1168
ql/PricingEngines/Quanto/ quantoengine.hpp (Quanto option engine)	1173
ql/PricingEngines/Swaption/ blackswaptionengine.hpp (Black-formula swaption engine)	1174
ql/PricingEngines/Swaption/ discretizedswaption.hpp (Discretized swaption class)	1175
ql/PricingEngines/Swaption/ g2swaptionengine.hpp (Swaption pricing engine for two-factor additive Gaussian Model G2++)	1176
ql/PricingEngines/Swaption/ jamshidianswaptionengine.hpp (Swaption engine using Jamshidian's decomposition)	1177
ql/PricingEngines/Swaption/ treeswaptionengine.hpp (Numerical lattice engine for swaptions)	1178
ql/PricingEngines/Vanilla/ analyticdigitalamericanengine.hpp (Analytic digital American option engine)	1179
ql/PricingEngines/Vanilla/ analyticdividendeuropeanengine.hpp (Analytic discrete-dividend European engine)	1180
ql/PricingEngines/Vanilla/ analyticeuropeanengine.hpp (Analytic European engine)	1181
ql/PricingEngines/Vanilla/ analytichestonengine.hpp (Analytic Heston-model engine)	1182
ql/PricingEngines/Vanilla/ baroneadesiwhaleyengine.hpp (Barone-Adesi and Whaley approximation engine)	1183
ql/PricingEngines/Vanilla/ batesengine.hpp (Analytic Bates model engine)	1184
ql/PricingEngines/Vanilla/ binomialengine.hpp (Binomial option engine)	1185
ql/PricingEngines/Vanilla/ bjerkhundstenslandengine.hpp (Bjerkhund and Stensland approximation engine)	1186
ql/PricingEngines/Vanilla/ discretizedvanillaoption.hpp (Discretized vanilla option)	1187
ql/PricingEngines/Vanilla/ fdamericanengine.hpp (Finite-differences American option engine)	1188
ql/PricingEngines/Vanilla/ fdbermudanengine.hpp (Finite-difference Bermudan engine)	1189
ql/PricingEngines/Vanilla/ fddividendamericanengine.hpp (American engine with discrete deterministic dividends)	1190
ql/PricingEngines/Vanilla/ fddividendengine.hpp (Base engine for option with dividends)	1191
ql/PricingEngines/Vanilla/ fddividendeuropeanengine.hpp (Finite-differences engine for European option with dividends)	1192
ql/PricingEngines/Vanilla/ fddividendshoutengine.hpp (Base class for shout engine with dividends)	1193
ql/PricingEngines/Vanilla/ fdeuropeanengine.hpp (Finite-difference European engine)	1194
ql/PricingEngines/Vanilla/ fdmultiengine.hpp (Base engine for options with events happening at specific times)	1195
ql/PricingEngines/Vanilla/ fdshoutengine.hpp (Finite-differences shout engine)	1196

ql/PricingEngines/Vanilla/ fdstepconditionengine.hpp (Finite-differences step-condition engine)	1197
ql/PricingEngines/Vanilla/ fdvanillaengine.hpp (Finite-differences vanilla-option engine)	1198
ql/PricingEngines/Vanilla/ integralengine.hpp (Integral option engine)	1199
ql/PricingEngines/Vanilla/ jumpdiffusionengine.hpp (Jump diffusion (Merton 1976) engine)	1200
ql/PricingEngines/Vanilla/ juquadraticengine.hpp (Ju quadratic (1999) approximation engine)	1201
ql/PricingEngines/Vanilla/ mcdigitalengine.hpp (Digital option Monte Carlo engine)	1202
ql/PricingEngines/Vanilla/ mceuropeanengine.hpp (Monte Carlo European option engine)	1203
ql/PricingEngines/Vanilla/ mceuropeanhestonengine.hpp (Monte Carlo Heston-model engine for European options)	1204
ql/PricingEngines/Vanilla/ mchestonengine.hpp (Monte Carlo Heston-model engine)	1205
ql/PricingEngines/Vanilla/ mcvanillaengine.hpp (Monte Carlo vanilla option engine)	1206
ql/Processes/ blackscholesprocess.hpp (Black-Scholes processes)	1207
ql/Processes/ capletlmmprocess.hpp (Stochastic process of a (cap) libor market model)	1208
ql/Processes/ defaultable.hpp (Defaultable processes)	1209
ql/Processes/ eulerdiscretization.hpp (Euler discretization for stochastic processes)	1210
ql/Processes/ geometricbrownianprocess.hpp (Geometric Brownian-motion process)	1211
ql/Processes/ hestonprocess.hpp (Heston stochastic process)	1212
ql/Processes/ merton76process.hpp (Merton-76 process)	1213
ql/Processes/ ornsteinuhlenbeckprocess.hpp (Ornstein-Uhlenbeck process)	1214
ql/Processes/ squarerootprocess.hpp (Square-root process)	1215
ql/Processes/ stochasticprocessarray.hpp (Array of correlated 1-D stochastic processes)	1216
ql/RandomNumbers/ boxmullergaussianrng.hpp (Box-Muller Gaussian random-number generator)	1220
ql/RandomNumbers/ centrallimitgaussianrng.hpp (Central limit Gaussian random-number generator)	1221
ql/RandomNumbers/ faurersg.hpp (Faure low-discrepancy sequence generator)	1222
ql/RandomNumbers/ haltonrsg.hpp (Halton low-discrepancy sequence generator)	1223
ql/RandomNumbers/ inversecumulativerng.hpp (Inverse cumulative Gaussian random-number generator)	1224
ql/RandomNumbers/ inversecumulativersg.hpp (Inverse cumulative random sequence generator)	1225
ql/RandomNumbers/ knuthuniformrng.hpp (Knuth uniform random number generator)	1226
ql/RandomNumbers/ lecuyeruniformrng.hpp (L'Ecuyer uniform random number generator)	1227
ql/RandomNumbers/ mt19937uniformrng.hpp (Mersenne Twister uniform random number generator)	1228
ql/RandomNumbers/ randomizedlds.hpp (Randomized low-discrepancy sequence)	1229
ql/RandomNumbers/ randomsequencegenerator.hpp (Random sequence generator based on a pseudo-random number generator)	1230
ql/RandomNumbers/ rngtraits.hpp (Random-number generation policies)	1231
ql/RandomNumbers/ seedgenerator.hpp (Random seed generator)	1233
ql/RandomNumbers/ sobolrsg.hpp (Sobol low-discrepancy sequence generator)	1234
ql/ShortRateModels/ calibrationhelper.hpp (Calibration helper class)	1237
ql/ShortRateModels/ model.hpp (Abstract interest rate model class)	1241
ql/ShortRateModels/ onefactormodel.hpp (Abstract one-factor interest rate model class)	1242
ql/ShortRateModels/ parameter.hpp (Model parameter classes)	1248
ql/ShortRateModels/ twofactormodel.hpp (Abstract two-factor interest rate model class)	1249
ql/ShortRateModels/CalibrationHelpers/ caphelper.hpp (CapHelper calibration helper)	1238
ql/ShortRateModels/CalibrationHelpers/ hestonmodelhelper.hpp (Heston-model calibration helper)	1239

ql/ShortRateModels/CalibrationHelpers/ swaptionhelper.hpp (Swaption calibration helper)	1240
ql/ShortRateModels/OneFactorModels/ blackkarasinski.hpp (Black-Karasinski model)	1243
ql/ShortRateModels/OneFactorModels/ coxingersollross.hpp (Cox-Ingersoll-Ross model)	1244
ql/ShortRateModels/OneFactorModels/ extendedcoxingersollross.hpp (Extended Cox-Ingersoll-Ross model)	1245
ql/ShortRateModels/OneFactorModels/ hullwhite.hpp (Hull & White (HW) model)	1246
ql/ShortRateModels/OneFactorModels/ vasicek.hpp (Vasicek model class)	1247
ql/ShortRateModels/TwoFactorModels/ batesmodel.hpp (Extended versions of the Heston model)	1250
ql/ShortRateModels/TwoFactorModels/ g2.hpp (Two-factor additive Gaussian Model G2++)	1251
ql/ShortRateModels/TwoFactorModels/ hestonmodel.hpp (Heston model for the stochastic volatility of an asset)	1252
ql/Solvers1D/ bisection.hpp (Bisection 1-D solver)	1254
ql/Solvers1D/ brent.hpp (Brent 1-D solver)	1255
ql/Solvers1D/ falseposition.hpp (False-position 1-D solver)	1256
ql/Solvers1D/ newton.hpp (Newton 1-D solver)	1257
ql/Solvers1D/ newtonsafe.hpp (Safe (bracketed) Newton 1-D solver)	1258
ql/Solvers1D/ ridder.hpp (Ridder 1-D solver)	1259
ql/Solvers1D/ secant.hpp (Secant 1-D solver)	1260
ql/TermStructures/ affinetermstructure.hpp (Affine term structure)	1264
ql/TermStructures/ bondhelpers.hpp (Bond rate helpers)	1265
ql/TermStructures/ bootstraptraits.hpp (Bootstrap traits)	1266
ql/TermStructures/ compoundforward.hpp (Compounded forward term structure)	1267
ql/TermStructures/ discountcurve.hpp (Interpolated discount factor structure)	1268
ql/TermStructures/ drifttermstructure.hpp (Drift term structure)	1269
ql/TermStructures/ extendeddiscountcurve.hpp (Discount factor structure with detailed compound-forward calculation)	1270
ql/TermStructures/ flatforward.hpp (Flat forward rate term structure)	1271
ql/TermStructures/ forwardcurve.hpp (Interpolated forward-rate structure)	1272
ql/TermStructures/ forwardspreadedtermstructure.hpp (Forward-spreaded term structure)	1273
ql/TermStructures/ forwardstructure.hpp (Forward-based yield term structure)	1274
ql/TermStructures/ impliedtermstructure.hpp (Implied term structure)	1275
ql/TermStructures/ piecewiseflatforward.hpp (Piecewise flat forward term structure)	1276
ql/TermStructures/ piecewiseyieldcurve.hpp (Piecewise-interpolated term structure)	1277
ql/TermStructures/ quantotermstructure.hpp (Quanto term structure)	1278
ql/TermStructures/ ratehelpers.hpp (Rate helpers base class)	1279
ql/TermStructures/ zerocurve.hpp (Interpolated zero-rates structure)	1280
ql/TermStructures/ zerospreadedtermstructure.hpp (Zero spreaded term structure)	1281
ql/TermStructures/ zeroyieldstructure.hpp (Zero-yield based term structure)	1282
ql/Utilities/ dataformatters.hpp (Output manipulators)	1286
ql/Utilities/ dataparsers.hpp (Classes used to parse data for input)	1287
ql/Utilities/ disposable.hpp (Generic disposable object with move semantics)	1288
ql/Utilities/ null.hpp (Null values)	1289
ql/Utilities/ observablevalue.hpp (Observable and assignable proxy to concrete value)	1290
ql/Utilities/ steppingiterator.hpp (Iterator advancing in constant steps)	1291
ql/Utilities/ strings.hpp (String utilities)	1292
ql/Utilities/ tracing.hpp (Tracing facilities)	1293
ql/Volatilities/ blackconstantvol.hpp (Black constant volatility, no time dependence, no strike dependence)	1295
ql/Volatilities/ blackvariancecurve.hpp (Black volatility curve modelled as variance curve)	1296

ql/Volatilities/ blackvariancesurface.hpp (Black volatility surface modelled as variance surface)	1297
ql/Volatilities/ capflatvolvector.hpp (Cap/floor at-the-money flat volatility vector)	1298
ql/Volatilities/ capletconstantvol.hpp (Constant caplet volatility)	1299
ql/Volatilities/ capletvariancecurve.hpp (Caplet variance curve)	1300
ql/Volatilities/ impliedvoltermstructure.hpp (Implied Black Vol Term Structure)	1301
ql/Volatilities/ localconstantvol.hpp (Local constant volatility, no time dependence, no asset dependence)	1302
ql/Volatilities/ localvolcurve.hpp (Local volatility curve derived from a Black curve) . .	1303
ql/Volatilities/ localvolsurface.hpp (Local volatility surface derived from a Black vol surface)	1304
ql/Volatilities/ swaptionvolmatrix.hpp (Swaption at-the-money volatility matrix)	1305

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::IMMMonth](#) { **H** = 3, **M** = 6, **U** = 9, **Z** = 12 }
Main cycle of the International Money Market (a.k.a. IMM) Months.
- 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](#), 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 [IMMMonth](#)

Main cycle of the International [Money](#) Market (a.k.a. [IMM](#)) Months.

Deprecated

use IMM::Month instead

6.3.2.4 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](#), 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 `Beijing`
Beijing calendar
- class `Bombay`
Bombay calendar
- class `Bratislava`
Bratislava calendar
- class `Budapest`
Budapest calendar
- class `Copenhagen`
Copenhagen calendar
- class `Germany`
German calendars.
- class `Helsinki`
Helsinki calendar
- class `HongKong`
Hong Kong calendar.
- class `Istanbul`
Istanbul calendar
- class `Italy`
Italian calendars.
- class `Johannesburg`
Johannesburg calendar
- class `JointCalendar`
Joint calendar.
- class `NullCalendar`
Calendar for reproducing theoretical calculations.

- class [Oslo](#)
Oslo calendar
- class [Prague](#)
Prague calendar
- class [Riyadh](#)
Riyadh calendar
- class [Seoul](#)
Seoul calendar
- class [Singapore](#)
Singapore calendar
- class [Stockholm](#)
Stockholm calendar
- class [Sydney](#)
Sydney calendar (New South Wales, Australia)
- class [Taipei](#)
Taipei calendar
- class [Taiwan](#)
Taiwan calendar
- class [TARGET](#)
TARGET calendar
- class [Tokyo](#)
Tokyo calendar
- class [Toronto](#)
Toronto calendar
- class [UnitedKingdom](#)
United Kingdom calendars.
- class [UnitedStates](#)
United States calendars.
- class [Warsaw](#)
Warsaw calendar
- class [Wellington](#)
Wellington calendar
- class [Zurich](#)
Zurich 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 [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 [BjerkstrandStenslandApproximationEngine](#)
- class [FDAmericanEngine](#)
Finite-differences pricing engine for American one asset options.
- class [FDBermudanEngine](#)
Finite-differences Bermudan engine.
- class [FDDividendAmericanEngine](#)
Finite-differences pricing engine for dividend American options.
- class [FDDividendEngine](#)
Base finite-differences pricing engine for dividend options.
- class [FDDividendEuropeanEngine](#)
Finite-differences pricing engine for dividend European options.
- class [FDDividendShoutEngine](#)
Finite-differences shout engine with dividends.
- class [FDEuropeanEngine](#)
Pricing engine for European options using finite-differences.
- class [FDShoutEngine](#)
Finite-differences pricing engine for shout vanilla options.
- class [FDStepConditionEngine](#)
Finite-differences pricing engine for American-style vanilla options.
- class [FDVanillaEngine](#)

Finite-differences pricing engine for BSM one asset 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 [MCHestonEngine](#)

Monte Carlo Heston-model engine.

- class [MCVanillaEngine](#)

Pricing engine for vanilla options using Monte Carlo simulation.

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

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

```

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

```

The grid upon which the model will act will be a logarithmic grid of underlying prices, i.e., f will be defined in a range $[\ln u_{\min}, \ln u_{\max}]$ discretized as an array $x_i, i = 0 \dots N - 1$ 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 [BSMTermOperator](#)
Black-Scholes-Merton differential operator.
- class [CrankNicolson](#)
Crank-Nicolson scheme for finite difference methods.
- class [DMinus](#)
 D_- matricial representation
- class [DPlus](#)
 D_+ matricial representation
- class [DPlusDMinus](#)
 D_+D_- matricial representation
- class [DZero](#)
 D_0 matricial representation
- class [ExplicitEuler](#)
Forward Euler scheme for finite difference methods.
- class [FiniteDifferenceModel](#)
Generic finite difference model.
- class [ImplicitEuler](#)
Backward Euler scheme for finite difference methods.
- class [MixedScheme](#)
Mixed (explicit/implicit) scheme for finite difference methods.
- class [OneFactorOperator](#)
Interest-rate single factor model differential operator.
- class [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.

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

The Black-Karasinski model

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

No analytical tractability here.

The extended Cox-Ingersoll-Ross model

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

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

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

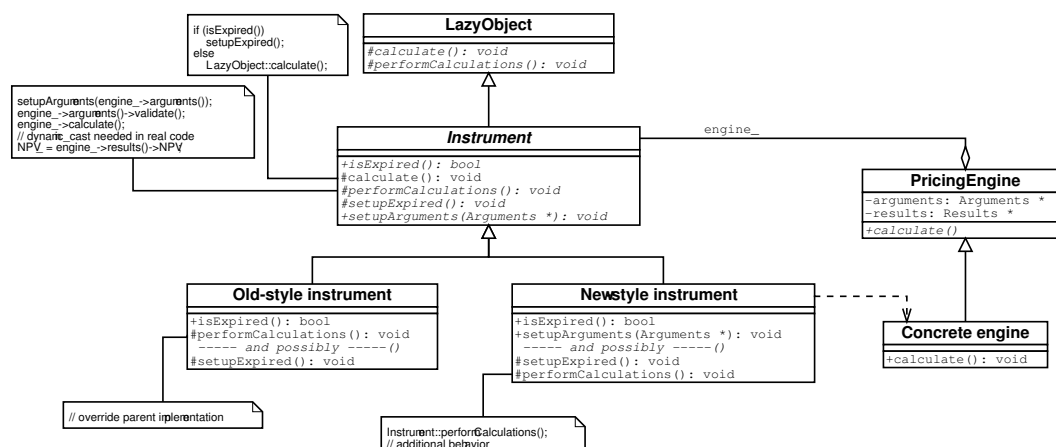
- class `AffineModel`
Affine model class.
- class `TermStructureConsistentModel`
Term-structure consistent model class.
- class `ShortRateModel`
Abstract short-rate model class.
- class `OneFactorModel`
Single-factor short-rate model abstract class.

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

The conjugate gradient method

We'll now continue with a bit more sophisticated method, implemented in [QuantLib::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

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

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

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

The [QuantLib::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)
    // just store the needed parameters
    : type_(type), underlying_(underlying), strike_(strike),
      discount_(discount), useAntithetic_(useAntithetic) {}
    // here is the logic
    double operator()(const Path& path) const {

        size_t n = path.size();

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

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

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

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

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

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

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

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

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

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

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

The path generator can be instantiated as

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

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

where `QuantLib::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 νT 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 "0.3.11"`
version string
- `#define QL_HEX_VERSION 0x000311f0`
version hexadecimal number
- `#define QL_LIB_VERSION "0_3_11"`
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:

[AmericanOption.cpp](#), [BermudanSwaption.cpp](#), [DiscreteHedging.cpp](#), [EuropeanOption.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](#).

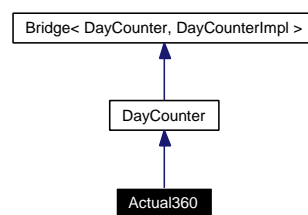
Chapter 7

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".

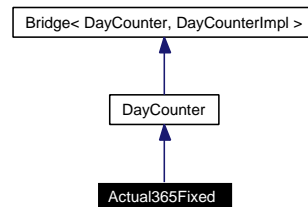
Examples:

[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.

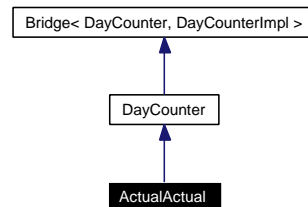
Examples:

[AmericanOption.cpp](#), [BermudanSwaption.cpp](#), [DiscreteHedging.cpp](#), and [EuropeanOption.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 }

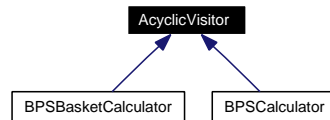
Public Member Functions

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

7.4 AcyclicVisitor Class Reference

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

Inheritance diagram for AcyclicVisitor:



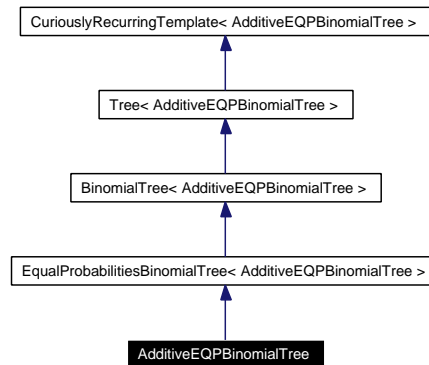
7.4.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

7.5 AdditiveEQPBinoomialTree Class Reference

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

Inheritance diagram for AdditiveEQPBinoomialTree:



7.5.1 Detailed Description

Additive equal probabilities binomial tree.

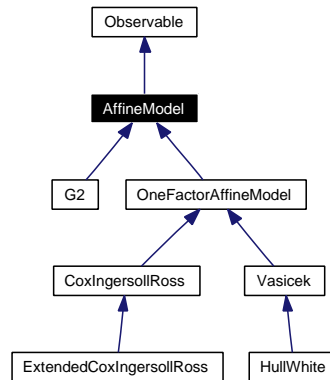
Public Member Functions

- **AdditiveEQPBinoomialTree** (const boost::shared_ptr< [StochasticProcess1D](#) > &process, [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.

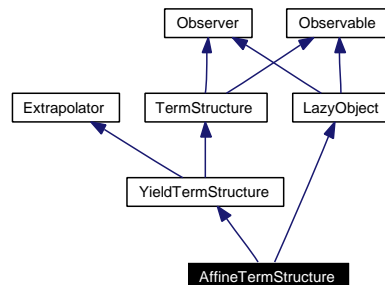
Public Member Functions

- virtual [DiscountFactor](#) `discount` ([Time](#) t) const =0
Implied discount curve.
- virtual [Real](#) `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]

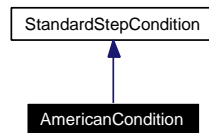
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [LazyObject](#).

7.8 AmericanCondition Class Reference

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

Inheritance diagram for AmericanCondition:



7.8.1 Detailed Description

American exercise condition.

Todo

unify the intrinsicValues/Payoff thing

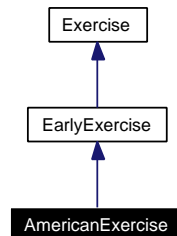
Public Member Functions

- **AmericanCondition** (Option::Type type, [Real](#) strike)
- **AmericanCondition** (const [Array](#) &intrinsicValues)
- void **applyTo** ([Array](#) &a, [Time](#) t) const

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:

[AmericanOption.cpp](#), and [EuropeanOption.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-Discout, [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

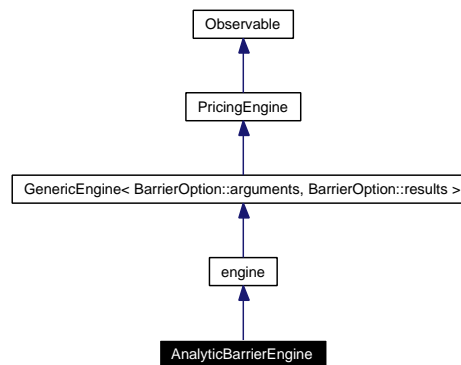
Public Member Functions

- **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

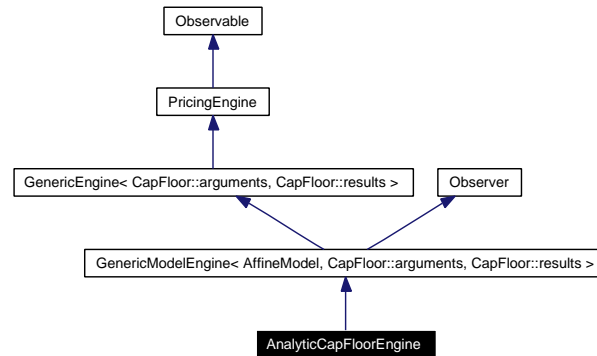
Public Member Functions

- 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.

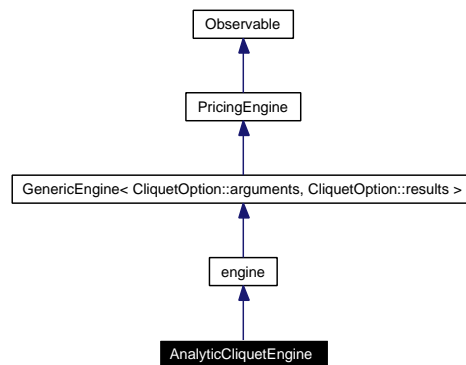
Public Member Functions

- **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.

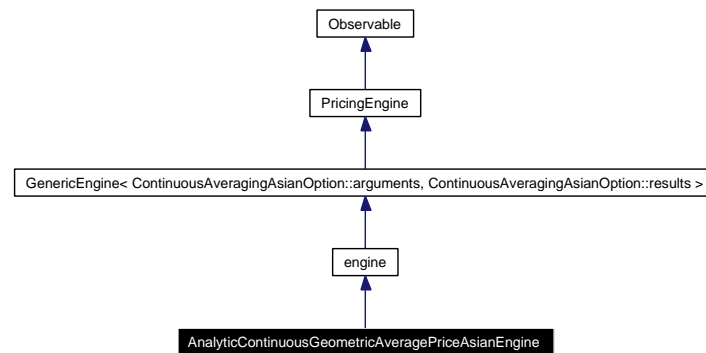
Public Member Functions

- 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

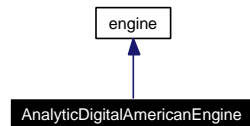
Public Member Functions

- 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.

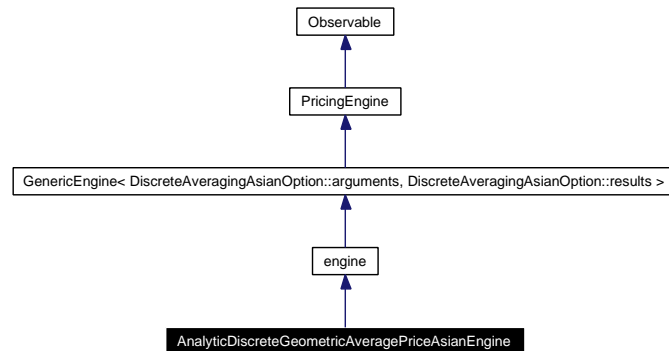
Public Member Functions

- 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.

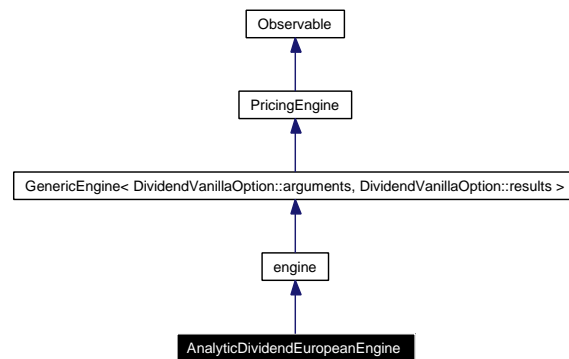
Public Member Functions

- 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.

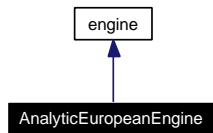
Public Member Functions

- 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:

[AmericanOption.cpp](#), and [EuropeanOption.cpp](#).

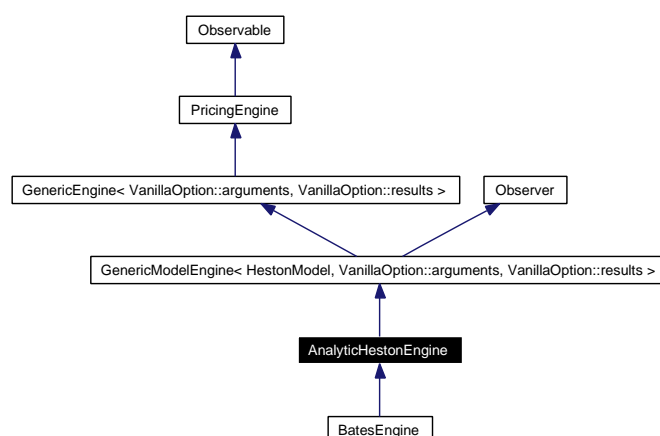
Public Member Functions

- 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.

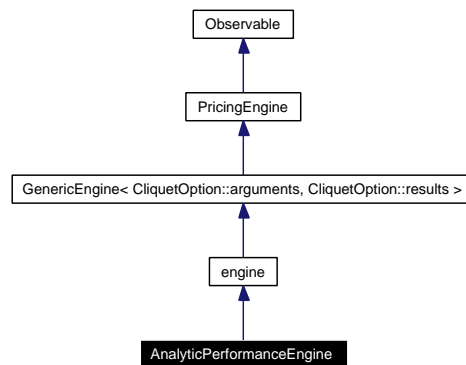
Public Member Functions

- **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

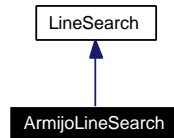
- void **calculate** () const

Inheritance diagram for Arguments:

7.23 ArmijoLineSearch Class Reference

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

Inheritance diagram for ArmijoLineSearch:



7.23.1 Detailed Description

Armijo line search.

Let α and β 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.

7.24 Array Class Reference

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

7.24.1 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.)

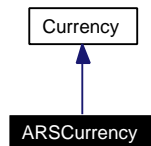
- [Real](#) **DotProduct** (const [Array](#) &, const [Array](#) &)
- const [Disposable](#)< [Array](#) > **operator**+ (const [Array](#) &v)
- const [Disposable](#)< [Array](#) > **operator**- (const [Array](#) &v)
- const [Disposable](#)< [Array](#) > **operator**+ (const [Array](#) &, const [Array](#)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

7.25 ARSCurrency Class Reference

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

Inheritance diagram for ARSCurrency:



7.25.1 Detailed Description

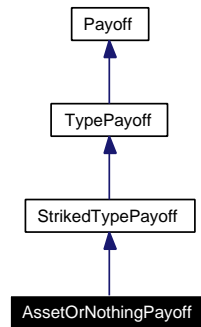
Argentinian peso.

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

7.26 AssetOrNothingPayoff Class Reference

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

Inheritance diagram for AssetOrNothingPayoff:



7.26.1 Detailed Description

Binary asset-or-nothing payoff.

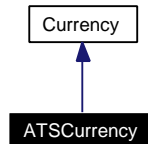
Public Member Functions

- **AssetOrNothingPayoff** (Option::Type type, [Real](#) strike)
- **[Real](#) operator()** ([Real](#) price) const

7.27 ATSCurrency Class Reference

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

Inheritance diagram for ATSCurrency:



7.27.1 Detailed Description

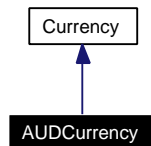
Austrian shilling.

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

7.28 AUDCurrency Class Reference

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

Inheritance diagram for AUDCurrency:



7.28.1 Detailed Description

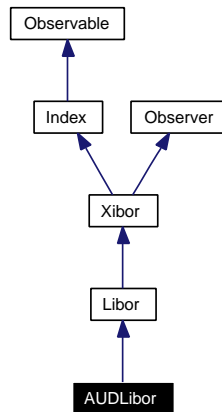
Australian dollar.

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

7.29 AUDLibor Class Reference

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

Inheritance diagram for AUDLibor:



7.29.1 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](#)())

7.30 Average Struct Reference

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

7.30.1 Detailed Description

placeholder for enumerated averaging types

Public Types

- enum Type { Arithmetic, Geometric }

7.31 BackwardFlat Class Reference

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

7.31.1 Detailed Description

Backward-flat interpolation factory and traits.

Public Types

- enum { **global** = 0 }

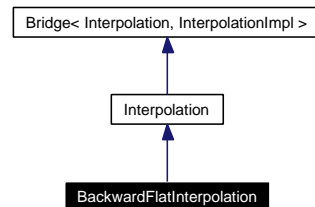
Public Member Functions

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

7.32 BackwardFlatInterpolation Class Reference

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

Inheritance diagram for BackwardFlatInterpolation:



7.32.1 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)`

7.32.2 Constructor & Destructor Documentation

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

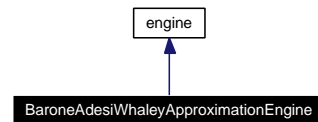
Precondition:

the x values must be sorted.

7.33 BaroneAdesiWhaleyApproximationEngine Class Reference

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

Inheritance diagram for BaroneAdesiWhaleyApproximationEngine:



7.33.1 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:

[AmericanOption.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)

7.34 Barrier Struct Reference

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

7.34.1 Detailed Description

Placeholder for enumerated barrier types.

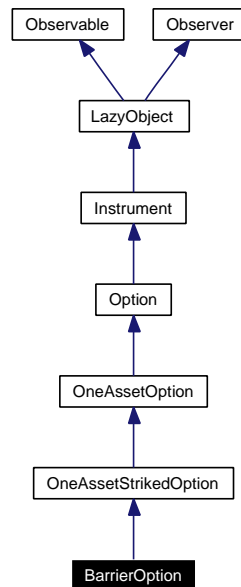
Public Types

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

7.35 BarrierOption Class Reference

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

Inheritance diagram for BarrierOption:



7.35.1 Detailed Description

Barrier option on a single asset.

The analytic pricing engine will be used if none is 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 Member Functions

- void **performCalculations** () const

Protected Attributes

- Barrier::Type **barrierType_**
- Real **barrier_**
- Real **rebate_**

Classes

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

7.35.2 Member Function Documentation

7.35.2.1 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](#).

7.35.2.2 void performCalculations () const [protected, virtual]

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

Reimplemented from [OneAssetStrikedOption](#).

7.36 BarrierOption::arguments Class Reference

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

7.36.1 Detailed Description

Arguments for barrier option calculation

Public Member Functions

- void **validate** () const

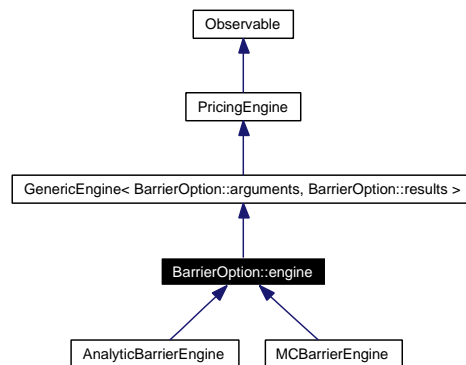
Public Attributes

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

7.37 BarrierOption::engine Class Reference

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

Inheritance diagram for BarrierOption::engine:



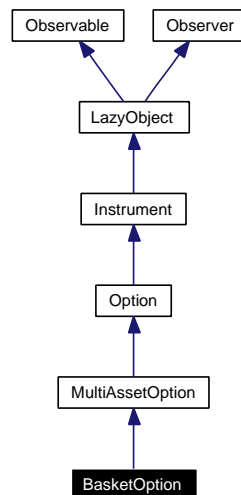
7.37.1 Detailed Description

Barrier engine base class

7.38 BasketOption Class Reference

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

Inheritance diagram for BasketOption:



7.38.1 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

7.38.2 Member Function Documentation

7.38.2.1 void setupArguments ([Arguments *](#)) const [virtual]

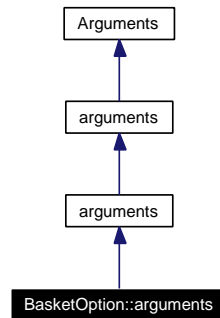
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [MultiAssetOption](#).

7.39 BasketOption::arguments Class Reference

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

Inheritance diagram for BasketOption::arguments:



7.39.1 Detailed Description

Arguments for basket option calculation

Public Member Functions

- void **validate** () const

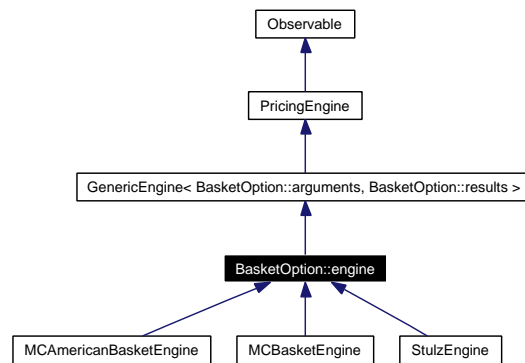
Public Attributes

- BasketType **basketType**

7.40 BasketOption::engine Class Reference

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

Inheritance diagram for BasketOption::engine:



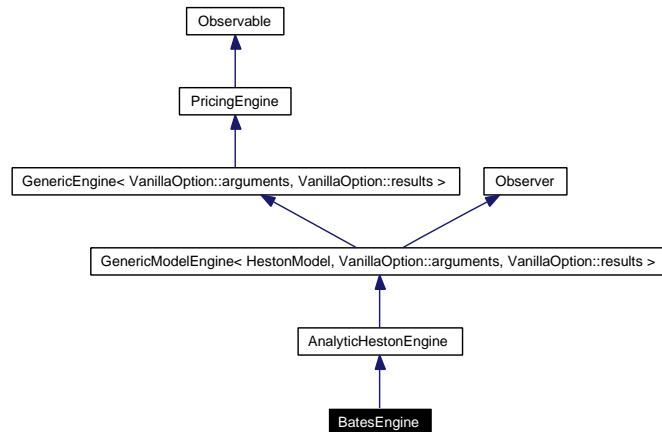
7.40.1 Detailed Description

Basket option engine base class

7.41 BatesEngine Class Reference

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

Inheritance diagram for BatesEngine:



7.41.1 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 λ . 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)

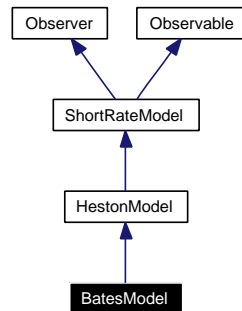
Protected Member Functions

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

7.42 BatesModel Class Reference

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

Inheritance diagram for BatesModel:



7.42.1 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.

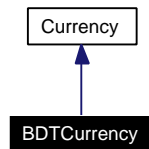
Public Member Functions

- **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

7.43 BDTCurrency Class Reference

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

Inheritance diagram for BDTCurrency:



7.43.1 Detailed Description

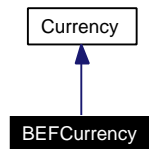
Bangladesh taka.

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

7.44 BEFCurrency Class Reference

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

Inheritance diagram for BEFCurrency:



7.44.1 Detailed Description

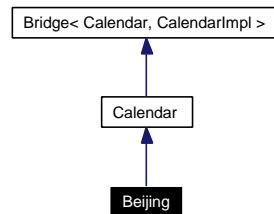
Belgian franc.

The ISO three-letter code is BEF; the numeric code is 56. It has no subdivisions.

7.45 Beijing Class Reference

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

Inheritance diagram for Beijing:



7.45.1 Detailed Description

Beijing calendar

Holidays:

- Saturdays
- Sundays
- New Year's day, January 1st
- Labour Day, first week in May
- National Day, one week from October 1st

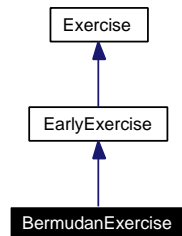
Other holidays for which no rule is given:

- Lunar New Year (data available for 2004 only)
- Spring Festival
- Last day of Lunar Year

7.46 BermudanExercise Class Reference

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

Inheritance diagram for BermudanExercise:



7.46.1 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:

[AmericanOption.cpp](#), [BermudanSwaption.cpp](#), and [EuropeanOption.cpp](#).

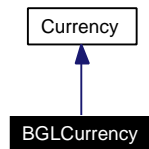
Public Member Functions

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

7.47 BGLCurrency Class Reference

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

Inheritance diagram for BGLCurrency:



7.47.1 Detailed Description

Bulgarian lev.

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

7.48 Bicubic Class Reference

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

7.48.1 Detailed Description

bicubic-spline interpolation factory

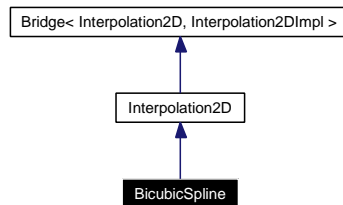
Public Member Functions

- `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`

7.49 BicubicSpline Class Reference

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

Inheritance diagram for BicubicSpline:



7.49.1 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)`

7.49.2 Constructor & Destructor Documentation

- 7.49.2.1 `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.

7.50 Bilinear Class Reference

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

7.50.1 Detailed Description

bilinear interpolation factory

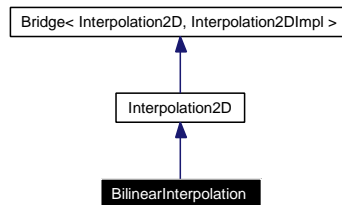
Public Member Functions

- `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`

7.51 BilinearInterpolation Class Reference

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

Inheritance diagram for BilinearInterpolation:



7.51.1 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)`

7.51.2 Constructor & Destructor Documentation

- 7.51.2.1 [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.

7.52 BinomialDistribution Class Reference

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

7.52.1 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.

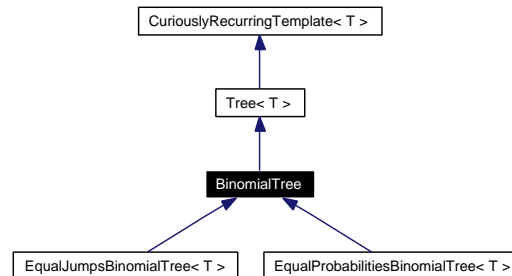
Public Member Functions

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

7.53 BinomialTree Class Template Reference

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

Inheritance diagram for BinomialTree:



7.53.1 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

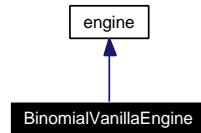
Protected Attributes

- [Real](#) x0_
- [Real](#) driftPerStep_
- [Time](#) dt_

7.54 BinomialVanillaEngine Class Template Reference

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

Inheritance diagram for BinomialVanillaEngine:



7.54.1 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.

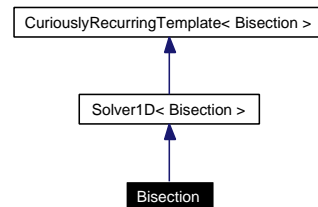
Public Member Functions

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

7.55 Bisection Class Reference

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

Inheritance diagram for Bisection:



7.55.1 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`

7.56 BivariateCumulativeNormalDistributionDr78 Class Reference

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

7.56.1 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

7.57 BivariateCumulativeNormalDistributionWe04DP Class Reference

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

7.57.1 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. 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.

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)

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.

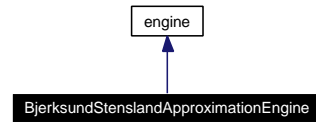
Public Member Functions

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

7.58 BjersundStenslandApproximationEngine Class Reference

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

Inheritance diagram for BjersundStenslandApproximationEngine:



7.58.1 Detailed Description

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

Tests

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

Examples:

[AmericanOption.cpp](#).

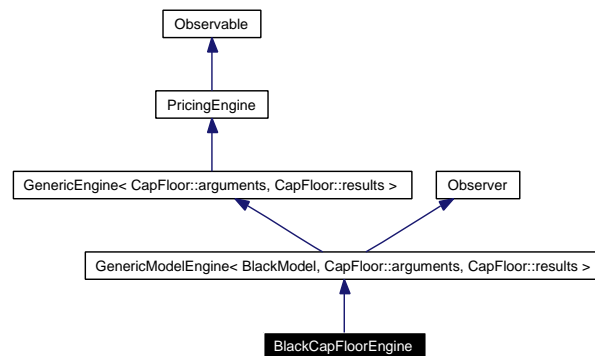
Public Member Functions

- void **calculate** () const

7.59 BlackCapFloorEngine Class Reference

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

Inheritance diagram for BlackCapFloorEngine:



7.59.1 Detailed Description

Black-formula cap/floor engine.

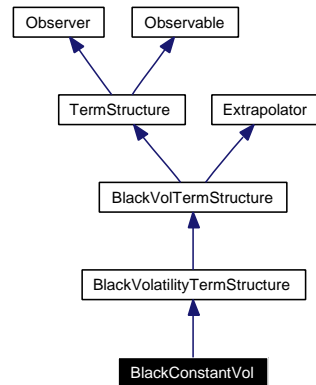
Public Member Functions

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

7.60 BlackConstantVol Class Reference

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

Inheritance diagram for BlackConstantVol:



7.60.1 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:

[AmericanOption.cpp](#), [DiscreteHedging.cpp](#), and [EuropeanOption.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.

7.61 BlackFormula Class Reference

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

7.61.1 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

7.61.2 Member Function Documentation

7.61.2.1 [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.

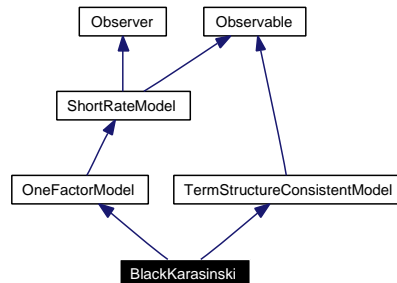
7.61.2.2 **Real** itmAssetProbability () const

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

7.62 BlackKarasinski Class Reference

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

Inheritance diagram for BlackKarasinski:



7.62.1 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.

7.63 BlackKarasinski::Dynamics Class Reference

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

7.63.1 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.

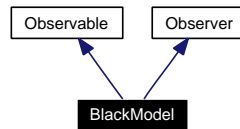
Public Member Functions

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

7.64 BlackModel Class Reference

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

Inheritance diagram for BlackModel:



7.64.1 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.

7.64.2 Member Function Documentation

7.64.2.1 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](#).

7.64.2.2 [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

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

7.64.2.3 **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

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

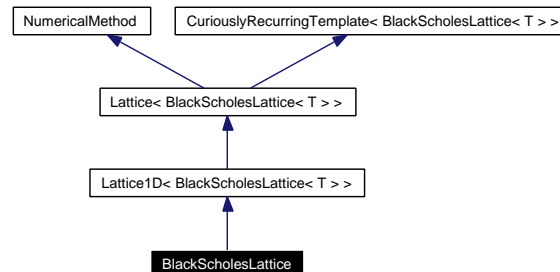
and

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

7.65 BlackScholesLattice Class Template Reference

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

Inheritance diagram for BlackScholesLattice:



7.65.1 Detailed Description

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

Simple binomial lattice approximating the Black-Scholes model.

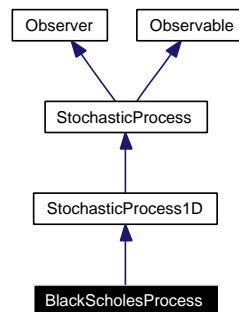
Public Member Functions

- **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

7.66 BlackScholesProcess Class Reference

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

Inheritance diagram for BlackScholesProcess:



7.66.1 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:

[AmericanOption.cpp](#), [DiscreteHedging.cpp](#), and [EuropeanOption.cpp](#).

Public Member Functions

- **BlackScholesProcess** (const [Handle< Quote >](#) &x0, const [Handle< YieldTermStructure >](#) ÷ndTS, 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`

7.66.2 Member Function Documentation

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

Todo

revise extrapolation

Implements [StochasticProcess1D](#).

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

Todo

revise extrapolation

Implements [StochasticProcess1D](#).

7.66.2.3 [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](#).

7.66.2.4 [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](#).

7.66.2.5 `void update ()` [virtual]

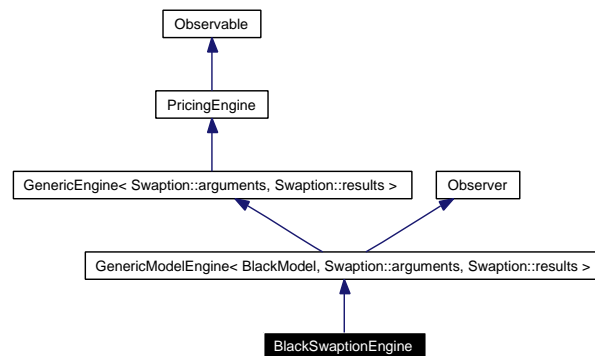
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [StochasticProcess](#).

7.67 BlackSwaptionEngine Class Reference

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

Inheritance diagram for BlackSwaptionEngine:



7.67.1 Detailed Description

Black-formula swaption engine.

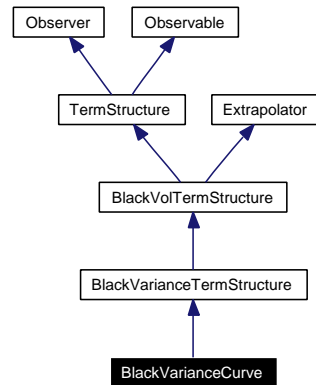
Public Member Functions

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

7.68 BlackVarianceCurve Class Reference

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

Inheritance diagram for BlackVarianceCurve:



7.68.1 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)

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](#) &)

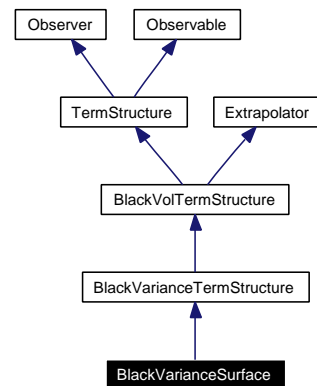
Protected Member Functions

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

7.69 BlackVarianceSurface Class Reference

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

Inheritance diagram for BlackVarianceSurface:



7.69.1 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

Examples:

[AmericanOption.cpp](#), and [EuropeanOption.cpp](#).

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 &)`

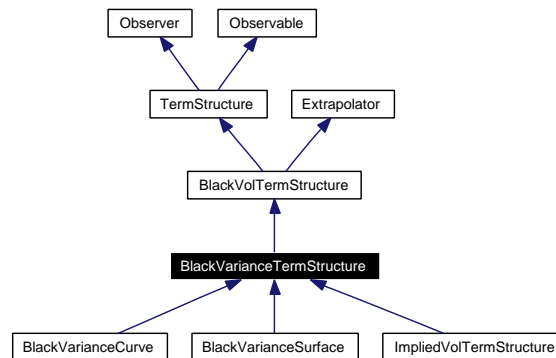
Protected Member Functions

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

7.70 BlackVarianceTermStructure Class Reference

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

Inheritance diagram for BlackVarianceTermStructure:



7.70.1 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

7.70.2 Constructor & Destructor Documentation

7.70.2.1 [BlackVarianceTermStructure](#) ()

default constructor

Warning:

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

7.70.3 Member Function Documentation

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

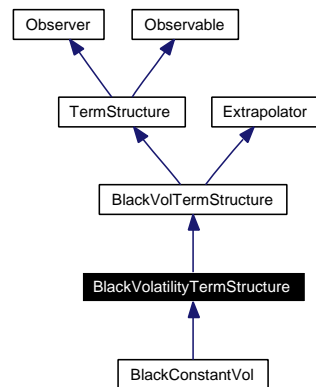
Returns the volatility for the given strike and date calculating it from the variance.

Implements [BlackVolTermStructure](#).

7.71 BlackVolatilityTermStructure Class Reference

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

Inheritance diagram for BlackVolatilityTermStructure:



7.71.1 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

7.71.2 Constructor & Destructor Documentation

7.71.2.1 [BlackVolatilityTermStructure](#) ()

default constructor

Warning:

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

7.71.3 Member Function Documentation

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

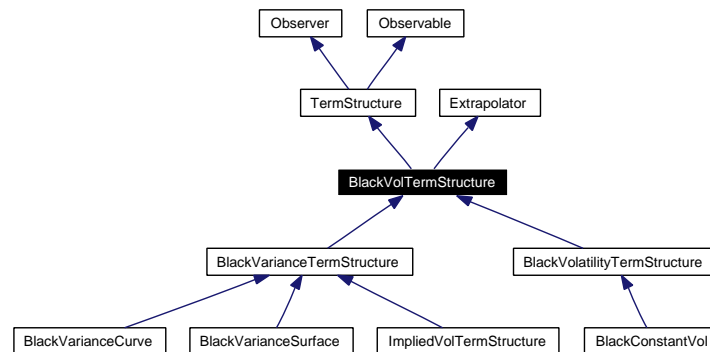
Returns the variance for the given strike and date calculating it from the volatility.

Implements [BlackVolTermStructure](#).

7.72 BlackVolTermStructure Class Reference

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

Inheritance diagram for BlackVolTermStructure:



7.72.1 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.

7.72.2 Constructor & Destructor Documentation

7.72.2.1 [BlackVolTermStructure](#) ()

default constructor

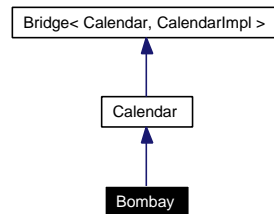
Warning:

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

7.73 Bombay Class Reference

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

Inheritance diagram for Bombay:



7.73.1 Detailed Description

Bombay calendar

Holidays (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

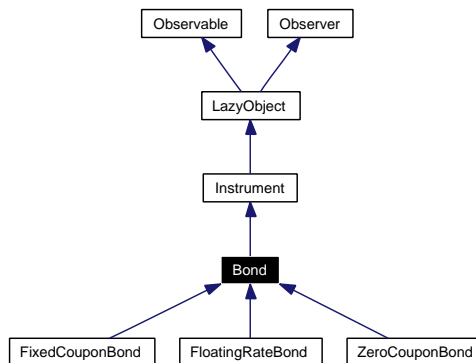
Other holidays for which no rule is given (data available for 2005 only:)

- Bakri Id
- Moharram
- Holi
- Maharashtra Day
- Ganesh Chaturthi
- Dasara
- Laxmi Puja
- Bhaubeej
- Ramzan Id
- Guru Nanak Jayanti

7.74 Bond Class Reference

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

Inheritance diagram for Bond:



7.74.1 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

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
- 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 maxEvaluations=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** >())
- **void** **performCalculations** () const

Protected Attributes

- **Integer** **settlementDays_**
- **Calendar** **calendar_**
- **BusinessDayConvention** **businessDayConvention_**
- **DayCounter** **dayCount_**
- **Date** **issueDate_**
- **Date** **datedDate_**
- **Date** **maturityDate_**
- **Frequency** **frequency_**
- **std::vector**< **boost::shared_ptr**< **CashFlow** > > **cashFlows_**
- **boost::shared_ptr**< **CashFlow** > **redemption_**
- **Handle**< **YieldTermStructure** > **discountCurve_**

7.74.2 Member Function Documentation

7.74.2.1 **Real** cleanPrice () const

theoretical clean price

The default bond settlement is used for calculation.

7.74.2.2 **Real** dirtyPrice () const

theoretical dirty price

The default bond settlement is used for calculation.

7.74.2.3 **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.

7.74.2.4 **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.

7.74.2.5 **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.

7.74.2.6 **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.

7.74.2.7 **Real** accruedAmount (**Date** *d* = Date()) const

accrued amount at a given date

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

7.74.2.8 **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](#).

7.75 BoundaryCondition Class Template Reference

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

7.75.1 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

7.75.2 Member Enumeration Documentation

7.75.2.1 enum [Side](#)

[Todo](#)

Generalize for n-dimensional conditions

7.75.3 Member Function Documentation

7.75.3.1 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](#).

7.75.3.2 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](#).

7.75.3.3 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](#).

7.75.3.4 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](#).

7.75.3.5 virtual void setTime (Time t) [pure virtual]

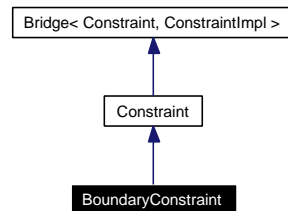
This method sets the current time for time-dependent boundary conditions.

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

7.76 BoundaryConstraint Class Reference

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

Inheritance diagram for BoundaryConstraint:



7.76.1 Detailed Description

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

Public Member Functions

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

7.77 BoxMullerGaussianRng Class Template Reference

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

7.77.1 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`

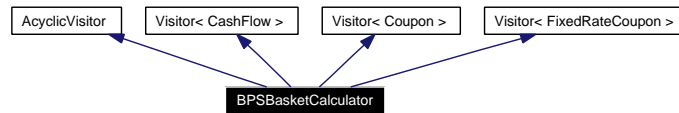
Public Member Functions

- `BoxMullerGaussianRng` (const RNG &uniformGenerator)
- `sample_type next` () const
returns a sample from a Gaussian distribution

7.78 BPSBasketCalculator Class Reference

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

Inheritance diagram for BPSBasketCalculator:



7.78.1 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

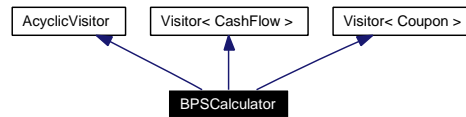
Visitor interface

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

7.79 BPSCalculator Class Reference

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

Inheritance diagram for BPSCalculator:



7.79.1 Detailed Description

basis point sensitivity (BPS) calculator

Instances of this class accumulate the BPS of each cash flow they visit, returning the sum through their result() method.

Public Member Functions

- **BPSCalculator** (const [Handle< YieldTermStructure >](#) &ts)
- [Real](#) **result** () const

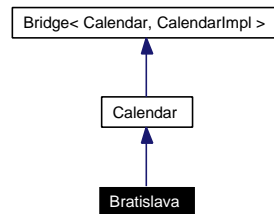
Visitor interface

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

7.80 Bratislava Class Reference

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

Inheritance diagram for Bratislava:



7.80.1 Detailed Description

Bratislava calendar

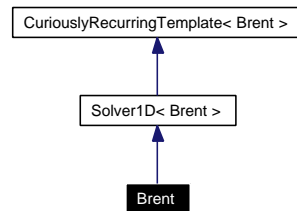
Holidays (see <http://www.bsse.sk/>):

- 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

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 >())

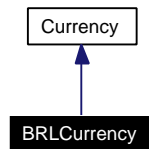
Protected Attributes

- 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)

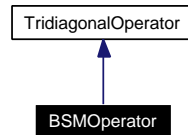
inspectors

- 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.

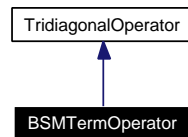
Public Member Functions

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

7.86 BSMTermOperator Class Reference

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

Inheritance diagram for BSMTermOperator:



7.86.1 Detailed Description

Black-Scholes-Merton differential operator.

Tests

coefficients are tested against constant BSM operator

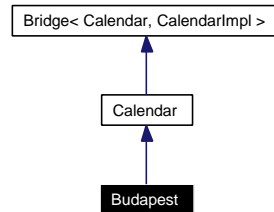
Public Member Functions

- **BSMTermOperator** (const [Array](#) &grid, const boost::shared_ptr< [BlackScholesProcess](#) > &, [Time](#) residualTime=0.0)

7.87 Budapest Class Reference

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

Inheritance diagram for Budapest:



7.87.1 Detailed Description

Budapest calendar

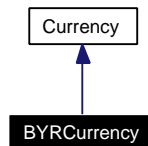
Holidays:

- Saturdays
- Sundays
- Easter Monday
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- National Day, March 15th
- Labour Day, May 1st
- Constitution Day, August 20th
- Republic Day, October 23rd
- All Saints Day, November 1st
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

7.88 BYRCurrency Class Reference

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

Inheritance diagram for BYRCurrency:



7.88.1 Detailed Description

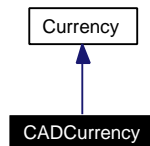
Belarussian ruble.

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

7.89 CADCurrency Class Reference

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

Inheritance diagram for CADCurrency:



7.89.1 Detailed Description

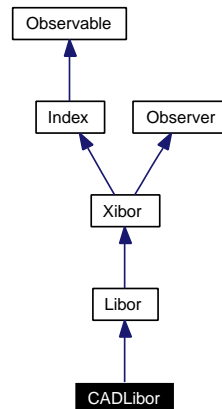
Canadian dollar.

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

7.90 CADLibor Class Reference

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

Inheritance diagram for CADLibor:



7.90.1 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.

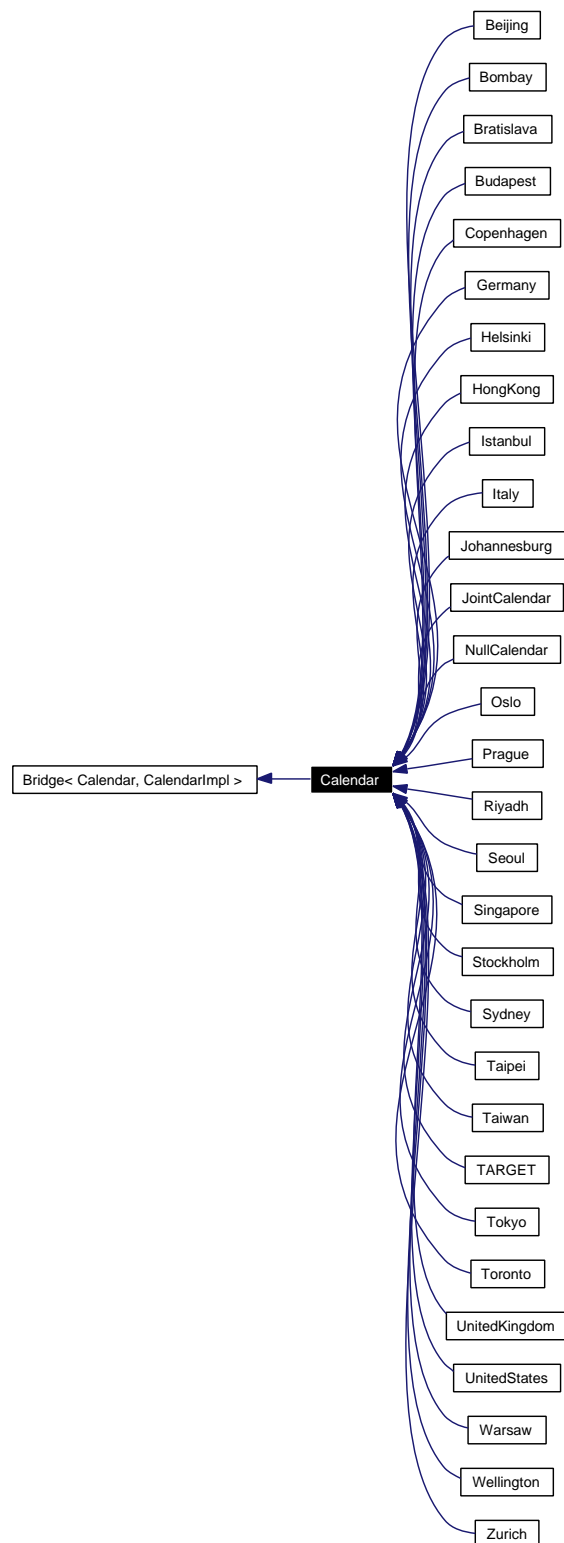
Public Member Functions

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

7.91 Calendar Class Reference

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

Inheritance diagram for Calendar:



7.91.1 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](#), 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 [WesternImpl](#)
partial calendar implementation

7.91.2 Constructor & Destructor Documentation

7.91.2.1 [Calendar](#) ()

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

7.91.2.2 [Calendar](#) (const boost::shared_ptr< [CalendarImpl](#) > & *impl*) [protected]

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

7.91.3 Member Function Documentation

7.91.3.1 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.

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

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

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

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

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

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

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

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

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

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

7.91.3.7 `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.

7.91.3.8 `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`.

7.91.3.9 `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.

7.91.4 Friends And Related Function Documentation

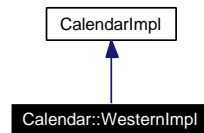
7.91.4.1 `bool` `operator==` (`const Calendar &`, `const Calendar &`) [related]

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

7.92 Calendar::WesternImpl Class Reference

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

Inheritance diagram for Calendar::WesternImpl:



7.92.1 Detailed Description

partial calendar implementation

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

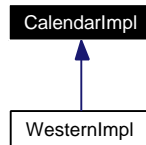
Static Protected Member Functions

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

7.93 CalendarImpl Class Reference

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

Inheritance diagram for CalendarImpl:



7.93.1 Detailed Description

abstract base class for calendar implementations

Public Member Functions

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

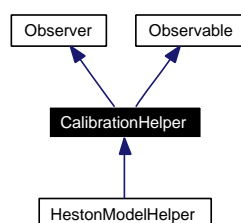
Public Attributes

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

7.94 CalibrationHelper Class Reference

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

Inheritance diagram for CalibrationHelper:



7.94.1 Detailed Description

liquid market instrument used during calibration

Public Member Functions

- **CalibrationHelper** (const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- 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](#) > ×) 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_](#)

7.94.2 Member Function Documentation

7.94.2.1 `void update ()` [virtual]

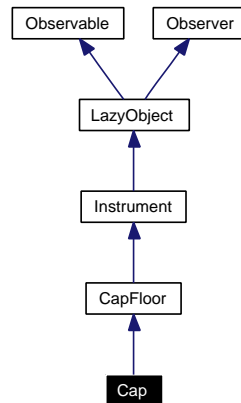
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.95 Cap Class Reference

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

Inheritance diagram for Cap:



7.95.1 Detailed Description

Concrete cap class.

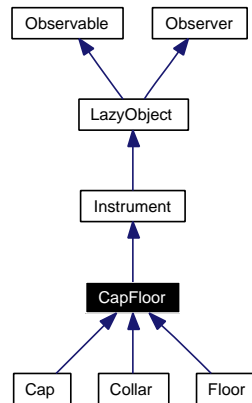
Public Member Functions

- `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]

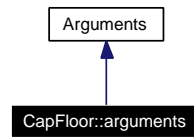
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

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

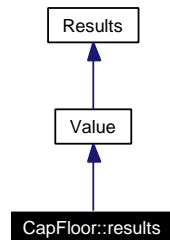
Public Attributes

- 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:



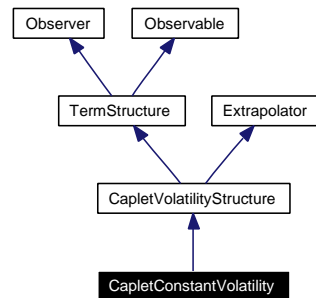
7.98.1 Detailed Description

Results from cap/floor calculation

7.99 CapletConstantVolatility Class Reference

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

Inheritance diagram for CapletConstantVolatility:



7.99.1 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

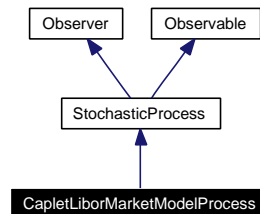
CapletVolatilityStructure interface

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

7.100 CapletLiborMarketModelProcess Class Reference

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

Inheritance diagram for CapletLiborMarketModelProcess:



7.100.1 Detailed Description

caplet libor-market-model process

stochastic process of a (cap) libor market 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.

Public Member Functions

- `CapletLiborMarketModelProcess` (`Size` fixings, `const boost::shared_ptr< Xibor >` &underlyingIndex, `const boost::shared_ptr< CapletVolatilityStructure >` &capletVol, `const Matrix` &volatilityComponents=`Matrix`())
- `Size size ()` `const`
returns the number of dimensions of the stochastic process
- `Size factors ()` `const`
returns the number of independent factors of the 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`
- `Disposable< Array > evolve (Time t0, const Array &x0, Time dt, const Array &dw) const`
- `std::vector< Time > fixingTimes () const`
- `Time accrualPeriod (Size i) const`
- `Volatility lambda (Size i, Size j=0) const`
- `DiscountFactor discountBond (const std::vector< Rate > &rates, Size j) const`
discount factor until the j -th fixing period

Protected Member Functions

- `Size nextResetDate (Time t) const`

7.100.2 Constructor & Destructor Documentation

- 7.100.2.1 `CapletLiborMarketModelProcess (Size fixings, const boost::shared_ptr< Xibor > &underlyingIndex, const boost::shared_ptr< CapletVolatilityStructure > &capletVol, const Matrix &volatilityComponents = Matrix())`

Parameters:

fixings number of rate fixing

underlyingIndex underlying `Libor` index

capletVol cap volatility term structure. Used to bootstrap volatilities Λ_i of F_i .

volatilityComponents $\lambda_{i,q}/\Lambda_i$, the ratio of the q -th component of the volatility of the forward rate to the total volatility of the forward rate. The number of columns of this matrix defines the number of factors of the model.

7.100.3 Member Function Documentation

- 7.100.3.1 `Disposable<Array> apply (const Array &x0, const Array &dx) const` [virtual]

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

Reimplemented from `StochasticProcess`.

- 7.100.3.2 `Disposable<Array> evolve (Time t0, const Array &x0, Time dt, const Array &dw) const` [virtual]

returns the asset value after a time interval Δt 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`.

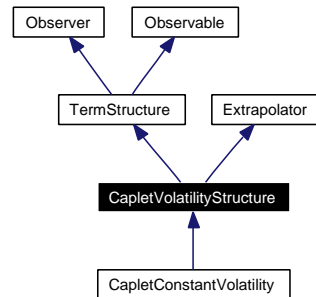
- 7.100.3.3 `Volatility lambda (Size i, Size j = 0) const`

volatility matrix $\lambda_{i,j}$, i -th fixing, j -th volatility factor, see equation 20 in Hull White paper

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

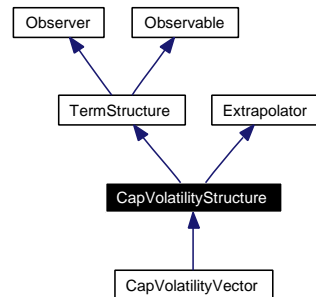
Warning:

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

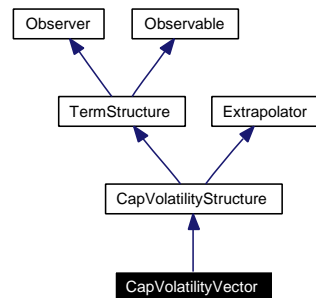
Warning:

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]

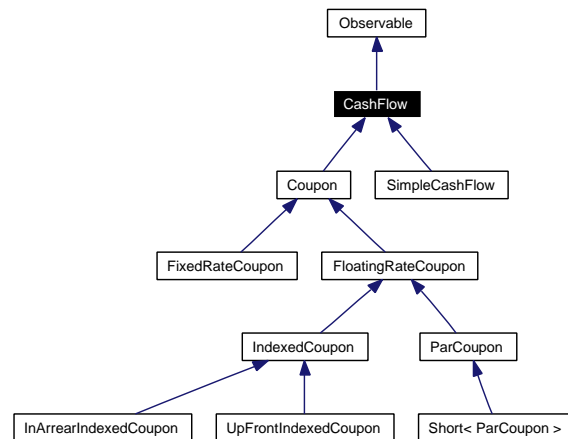
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

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
returns the date at which the cash flow is settled

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 [FixedRateCoupon](#), [IndexedCoupon](#), [ParCoupon](#), [Short< ParCoupon >](#), 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 [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)
Internal rate of return.
- static [Real](#) [convexity](#) (const std::vector< boost::shared_ptr< [CashFlow](#) > > &, const [InterestRate](#) &, [Date](#) settlementDate=[Date](#)())
Cash-flow convexity.
- static [Time](#) [duration](#) (const std::vector< boost::shared_ptr< [CashFlow](#) > > &, [Real](#) marketPrice, const [InterestRate](#) &, [Duration::Type](#) type=[Duration::Simple](#), [Date](#) settlementDate=[Date](#)())
Cash-flow duration.

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 **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.

7.105.2.4 static **Real** convexity (const std::vector< boost::shared_ptr< **CashFlow** > > &, const **InterestRate** &, **Date** settlementDate = **Date**()) [static]

Cash-flow convexity.

The convexity is defined as

$$C = \sum t^2 c_t P_t$$

where c_t is the amount of the cash flow and P_t is the discount at time t as implied by the given interest rate.

7.105.2.5 static **Time** duration (const std::vector< boost::shared_ptr< **CashFlow** > > &, **Real** marketPrice, const **InterestRate** &, **Duration::Type** type = **Duration::Simple**, **Date** settlementDate = **Date**()) [static]

Cash-flow duration.

The simple duration is defined as

$$D_{\text{simple}} = \frac{\sum t c_t P_t}{\sum c_t P_t}$$

where c_t is the amount of the cash flow and P_t is the discount at time t as implied by the given interest rate.

The modified duration is

$$D_{\text{modified}} = \frac{1}{y} D_{\text{simple}}$$

where y is the IRR.

Finally, the Macaulay duration is

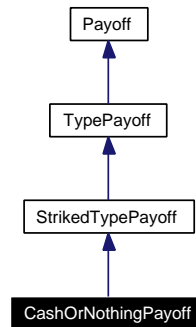
$$D_{\text{Macaulay}} = \frac{\sum t c_t P'_t}{\sum c_t P'_t}$$

where $P'_t = e^{-yt}$ and y is the IRR.

7.106 CashOrNothingPayoff Class Reference

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

Inheritance diagram for CashOrNothingPayoff:



7.106.1 Detailed Description

Binary cash-or-nothing payoff.

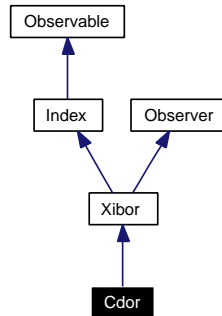
Public Member Functions

- **CashOrNothingPayoff** (Option::Type type, [Real](#) strike, [Real](#) cashPayoff)
- [Real](#) operator() ([Real](#) price) const
- [Real](#) cashPayoff () const

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.

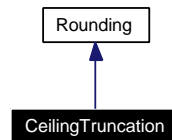
Public Member Functions

- **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.

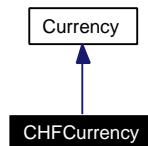
Public Member Functions

- **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

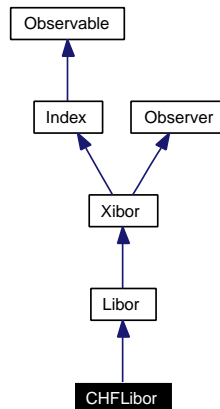
Swiss franc.

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 CLGaussianRng Class Template Reference

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

7.111.1 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`

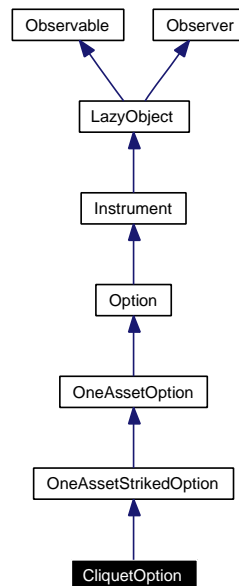
Public Member Functions

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

7.112 CliquetOption Class Reference

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

Inheritance diagram for CliquetOption:



7.112.1 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.

7.112.2 Member Function Documentation

7.112.2.1 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](#).

7.113 CliquetOption::arguments Class Reference

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

7.113.1 Detailed Description

Arguments for cliquet option calculation

Public Member Functions

- void **validate** () const

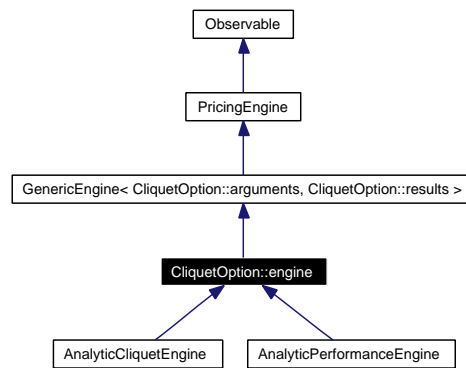
Public Attributes

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

7.114 CliquetOption::engine Class Reference

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

Inheritance diagram for CliquetOption::engine:



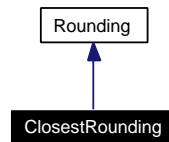
7.114.1 Detailed Description

Cliquet engine base class.

7.115 ClosestRounding Class Reference

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

Inheritance diagram for ClosestRounding:



7.115.1 Detailed Description

Closest rounding.

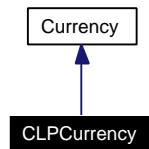
Public Member Functions

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

7.116 CLPCurrency Class Reference

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

Inheritance diagram for CLPCurrency:



7.116.1 Detailed Description

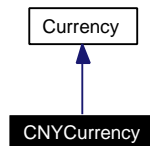
Chilean peso.

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

7.117 CNYCurrency Class Reference

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

Inheritance diagram for CNYCurrency:



7.117.1 Detailed Description

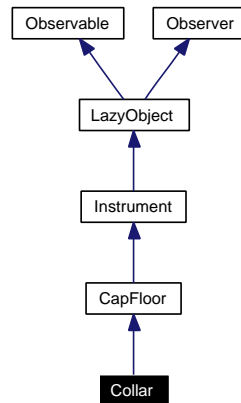
Chinese yuan.

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

7.118 Collar Class Reference

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

Inheritance diagram for Collar:



7.118.1 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)

7.119 Composite Class Template Reference

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

7.119.1 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)

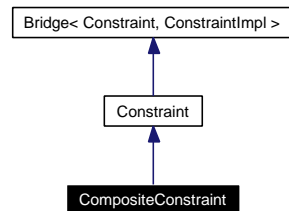
Protected Attributes

- std::list< boost::shared_ptr< T > > **components_**

7.120 CompositeConstraint Class Reference

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

Inheritance diagram for CompositeConstraint:



7.120.1 Detailed Description

Constraint enforcing both given sub-constraints

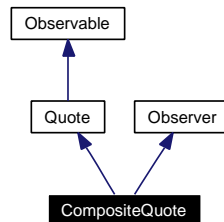
Public Member Functions

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

7.121 CompositeQuote Class Template Reference

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

Inheritance diagram for CompositeQuote:



7.121.1 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](#) ()

7.121.2 Member Function Documentation

7.121.2.1 void update () [virtual]

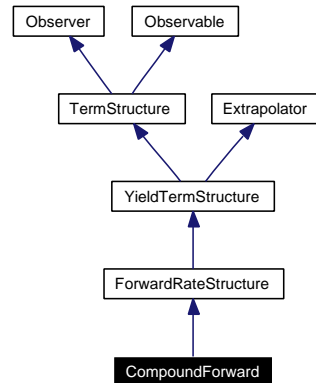
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.122 CompoundForward Class Reference

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

Inheritance diagram for CompoundForward:



7.122.1 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`

7.122.2 Member Function Documentation

7.122.2.1 [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](#).

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

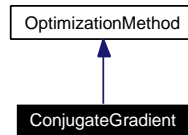
Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Reimplemented from [ForwardRateStructure](#).

7.123 ConjugateGradient Class Reference

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

Inheritance diagram for ConjugateGradient:



7.123.1 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 $d_1 = -f'(x_1)$

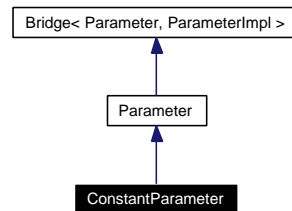
Public Member Functions

- [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

7.124 ConstantParameter Class Reference

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

Inheritance diagram for ConstantParameter:



7.124.1 Detailed Description

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

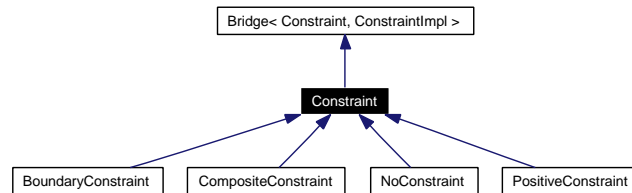
Public Member Functions

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

7.125 Constraint Class Reference

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

Inheritance diagram for Constraint:



7.125.1 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](#) >())

7.126 ConstraintImpl Class Reference

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

7.126.1 Detailed Description

Base class for constraint implementations.

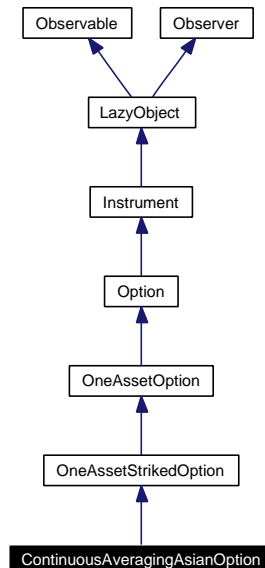
Public Member Functions

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

7.127 ContinuousAveragingAsianOption Class Reference

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

Inheritance diagram for ContinuousAveragingAsianOption:



7.127.1 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.

7.127.2 Member Function Documentation

7.127.2.1 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](#).

7.128 ContinuousAveragingAsianOption::arguments Class Reference

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

7.128.1 Detailed Description

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

Public Member Functions

- void **validate** () const

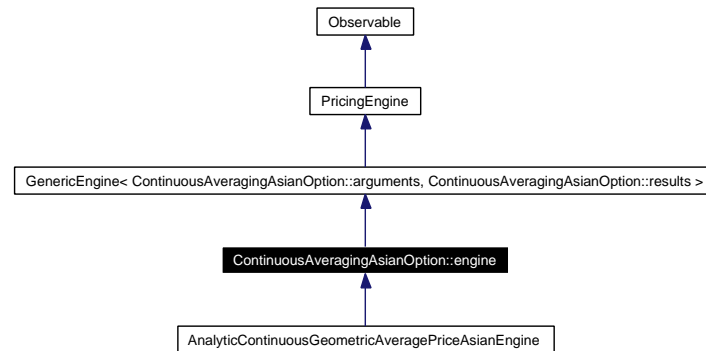
Public Attributes

- Average::Type **averageType**

7.129 ContinuousAveragingAsianOption::engine Class Reference

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

Inheritance diagram for ContinuousAveragingAsianOption::engine:



7.129.1 Detailed Description

Continuous-averaging Asian engine base class.

7.130 ConvergenceStatistics Class Template Reference

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

7.130.1 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.

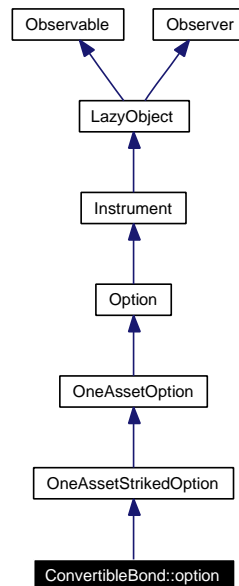
Public Member Functions

- **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

7.131 ConvertibleBond::option Class Reference

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

Inheritance diagram for ConvertibleBond::option:



7.131.1 Detailed Description

[Option](#) like features for Convertible [Bond](#) calculation.

Public Member Functions

- void [setupArguments](#) ([Arguments](#) *) const

Classes

- class [arguments](#)
Arguments for Convertible [Bond](#) calculation
- class [engine](#)
convertible bond engine base class

7.131.2 Member Function Documentation

7.131.2.1 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](#).

7.132 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

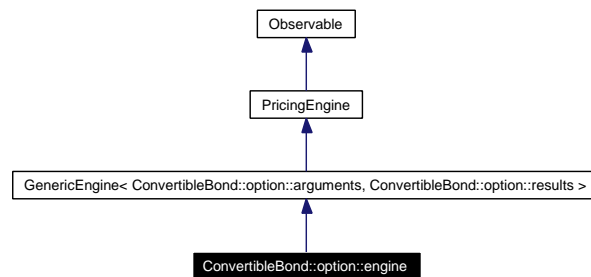
Public Attributes

- [Real](#) **conversionRatio**
- DividendSchedule **dividends**
- CallabilitySchedule **callability**
- [Handle](#)< [Quote](#) > **creditSpread**

7.133 ConvertibleBond::option::engine Class Reference

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

Inheritance diagram for ConvertibleBond::option::engine:



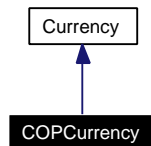
7.133.1 Detailed Description

convertible bond engine base class

7.134 COPCurrency Class Reference

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

Inheritance diagram for COPCurrency:



7.134.1 Detailed Description

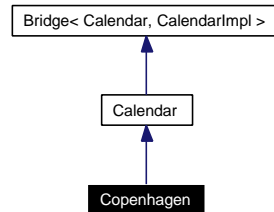
Colombian peso.

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

7.135 Copenhagen Class Reference

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

Inheritance diagram for Copenhagen:



7.135.1 Detailed Description

Copenhagen calendar

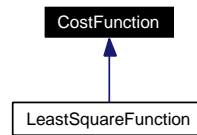
Holidays:

- 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

7.136 CostFunction Class Reference

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

Inheritance diagram for CostFunction:



7.136.1 Detailed Description

Cost function abstract class for optimization problem.

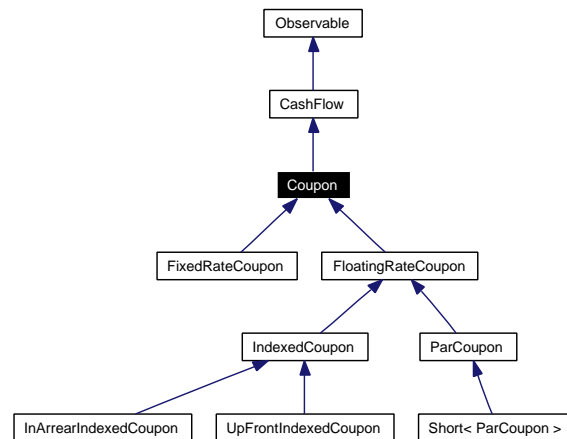
Public Member Functions

- virtual [Real value](#) (const [Array](#) &x) const =0
method to overload to compute the cost function value in x
- 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 .:

7.137 Coupon Class Reference

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

Inheritance diagram for Coupon:



7.137.1 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
returns the date at which the cash flow is settled

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_`

7.137.2 Constructor & Destructor Documentation

7.137.2.1 `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.

7.138 CovarianceDecomposition Class Reference

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

7.138.1 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

7.138.2 Constructor & Destructor Documentation

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

Precondition:

covarianceMatrix must be symmetric

7.138.3 Member Function Documentation

7.138.3.1 const [Array](#)& [variances](#) () const

returns the variances [Array](#)

7.138.3.2 const [Array](#)& [standardDeviations](#) () const

returns the standard deviations [Array](#)

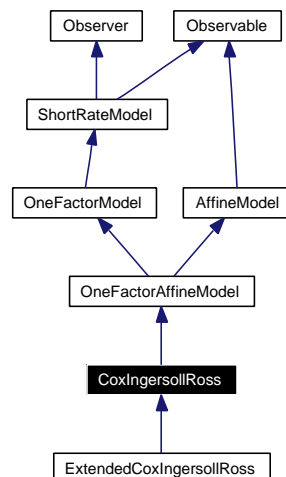
7.138.3.3 const [Matrix](#)& [correlationMatrix](#) () const

returns the correlation matrix

7.139 CoxIngersollRoss Class Reference

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

Inheritance diagram for CoxIngersollRoss:



7.139.1 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

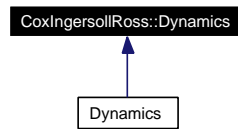
Classes

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

7.140 CoxIngersollRoss::Dynamics Class Reference

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

Inheritance diagram for CoxIngersollRoss::Dynamics:



7.140.1 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

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

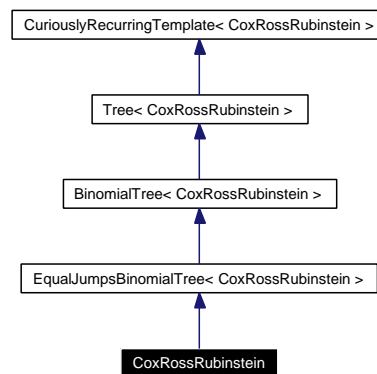
Public Member Functions

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

7.141 CoxRossRubinstein Class Reference

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

Inheritance diagram for CoxRossRubinstein:



7.141.1 Detailed Description

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

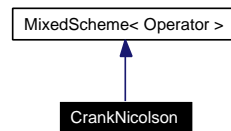
Public Member Functions

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

7.142 CrankNicolson Class Template Reference

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

Inheritance diagram for CrankNicolson:



7.142.1 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)

7.143 Cubic Class Reference

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

7.143.1 Detailed Description

cubic-spline interpolation factory and traits

Public Types

- enum { **global** = 1 }

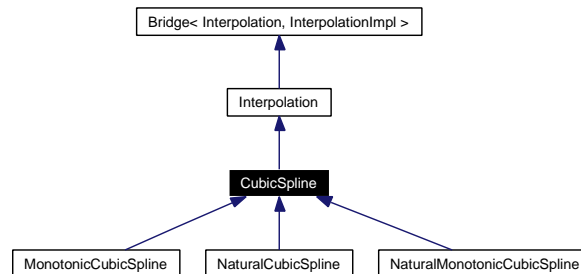
Public Member Functions

- **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

7.144 CubicSpline Class Reference

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

Inheritance diagram for CubicSpline:



7.144.1 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

7.144.2 Member Enumeration Documentation

7.144.2.1 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

7.144.3 Constructor & Destructor Documentation

7.144.3.1 [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.

7.145 CumulativeBinomialDistribution Class Reference

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

7.145.1 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

7.146 CumulativeNormalDistribution Class Reference

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

7.146.1 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

7.147 CumulativePoissonDistribution Class Reference

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

7.147.1 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.

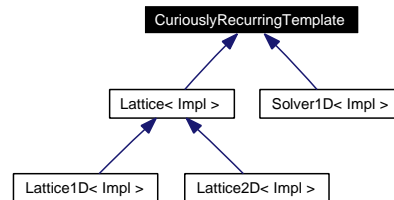
Public Member Functions

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

7.148 CuriouslyRecurringTemplate Class Template Reference

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

Inheritance diagram for CuriouslyRecurringTemplate:



7.148.1 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.

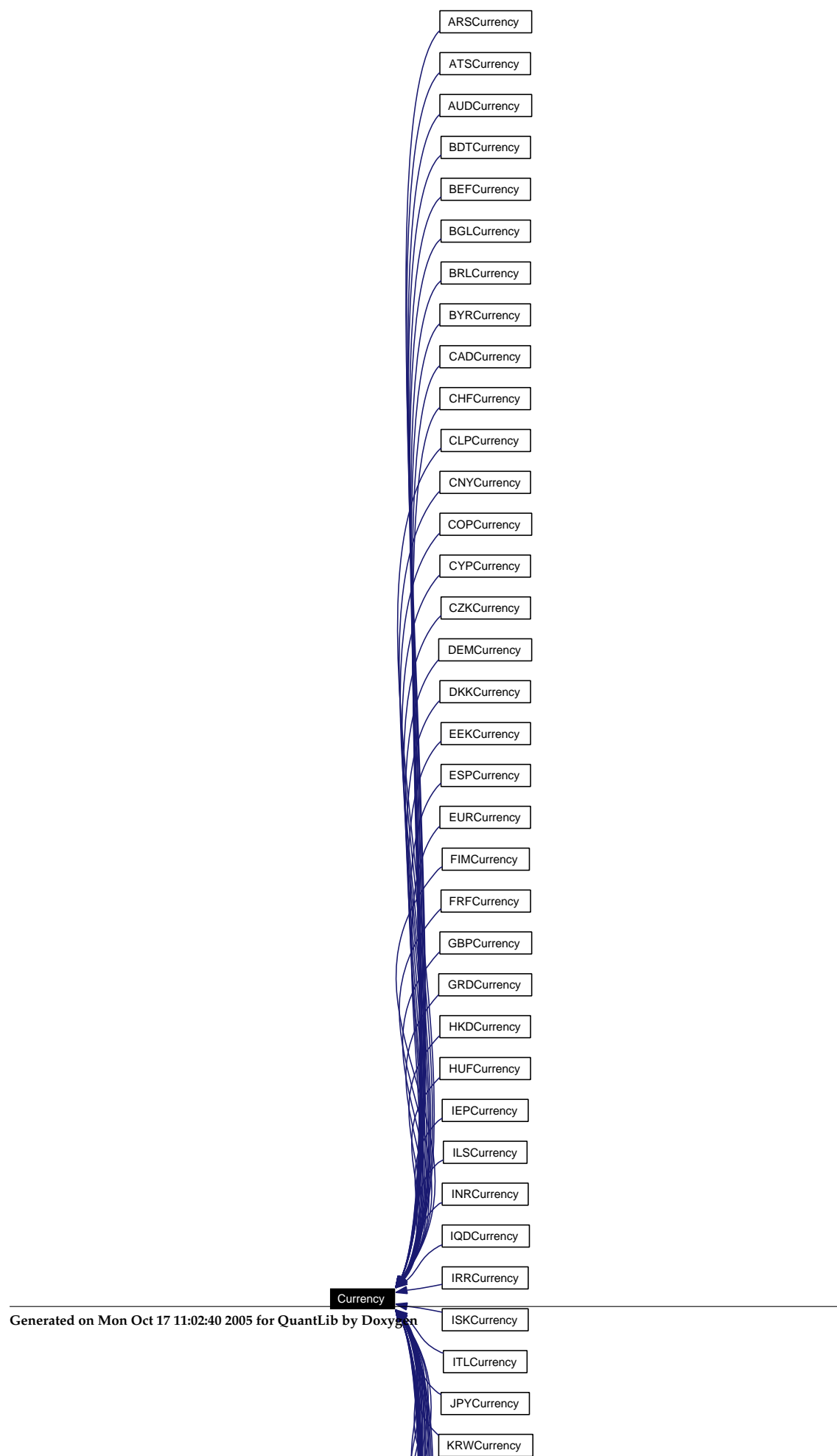
Protected Member Functions

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

7.149 Currency Class Reference

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

Inheritance diagram for Currency:



7.149.1 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](#) &)

7.149.2 Constructor & Destructor Documentation

7.149.2.1 [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.

7.149.3 Member Function Documentation

7.149.3.1 `std::string format () const`

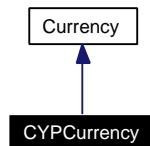
output format

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

7.150 CYPCurrency Class Reference

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

Inheritance diagram for CYPCurrency:



7.150.1 Detailed Description

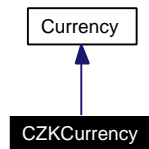
Cyprus pound.

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

7.151 CZKCurrency Class Reference

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

Inheritance diagram for CZKCurrency:



7.151.1 Detailed Description

Czech koruna.

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

7.152 Date Class Reference

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

7.152.1 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:

[AmericanOption.cpp](#), [BermudanSwaption.cpp](#), [DiscreteHedging.cpp](#), [EuropeanOption.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](#) &)

7.152.2 Member Function Documentation

7.152.2.1 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>

7.152.2.2 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>

7.152.2.3 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.

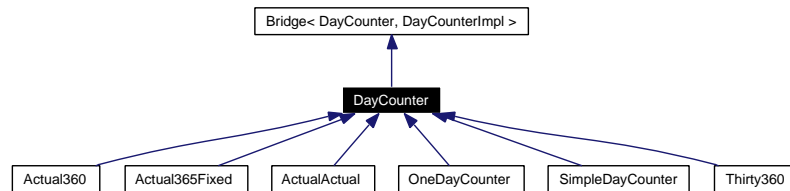
Warning:

The result date is following or equal to the original date

7.153 DayCounter Class Reference

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

Inheritance diagram for DayCounter:



7.153.1 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:

[AmericanOption.cpp](#), [BermudanSwaption.cpp](#), [DiscreteHedging.cpp](#), [EuropeanOption.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 &)`

7.153.2 Constructor & Destructor Documentation

7.153.2.1 DayCounter ()

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

7.153.2.2 DayCounter (const boost::shared_ptr< DayCounterImpl > & impl) [protected]

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

7.153.3 Member Function Documentation

7.153.3.1 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.

7.153.4 Friends And Related Function Documentation

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

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

7.154 DayCounterImpl Class Reference

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

7.154.1 Detailed Description

abstract base class for day counter implementations

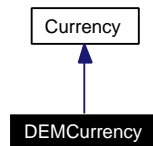
Public Member Functions

- 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

7.155 DEMCurrency Class Reference

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

Inheritance diagram for DEMCurrency:



7.155.1 Detailed Description

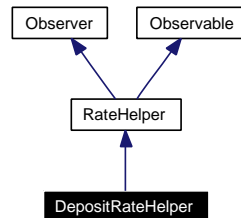
Deutsche mark.

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

7.156 DepositRateHelper Class Reference

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

Inheritance diagram for DepositRateHelper:



7.156.1 Detailed Description

Deposit rate.

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

7.156.2 Member Function Documentation

7.156.2.1 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](#).

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

latest relevant date

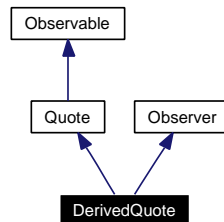
The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

7.157 DerivedQuote Class Template Reference

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

Inheritance diagram for DerivedQuote:



7.157.1 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 ()`

7.157.2 Member Function Documentation

7.157.2.1 void update () [virtual]

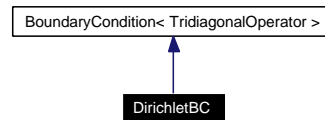
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.158 DirichletBC Class Reference

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

Inheritance diagram for DirichletBC:



7.158.1 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](#))

7.158.2 Member Function Documentation

7.158.2.1 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 >](#).

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

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

7.158.2.3 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 >](#).

7.158.2.4 void applyAfterSolving ([Array](#) &) const [virtual]

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

7.158.2.5 void setTime ([Time](#)) [virtual]

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

7.159 Discount Struct Reference

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

7.159.1 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)

7.160 DiscrepancyStatistics Class Reference

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

Inheritance diagram for DiscrepancyStatistics:



7.160.1 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)`

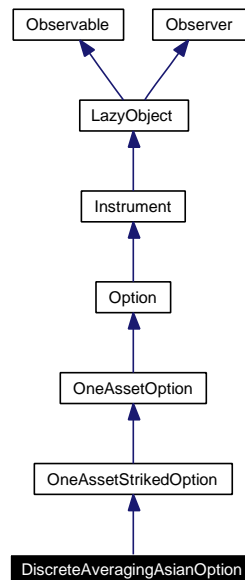
1-dimensional inspectors

- `Real discrepancy () const`

7.161 DiscreteAveragingAsianOption Class Reference

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

Inheritance diagram for DiscreteAveragingAsianOption:



7.161.1 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.

7.161.2 Member Function Documentation

7.161.2.1 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](#).

7.162 DiscreteAveragingAsianOption::arguments Class Reference

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

7.162.1 Detailed Description

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

Public Member Functions

- void **validate** () const

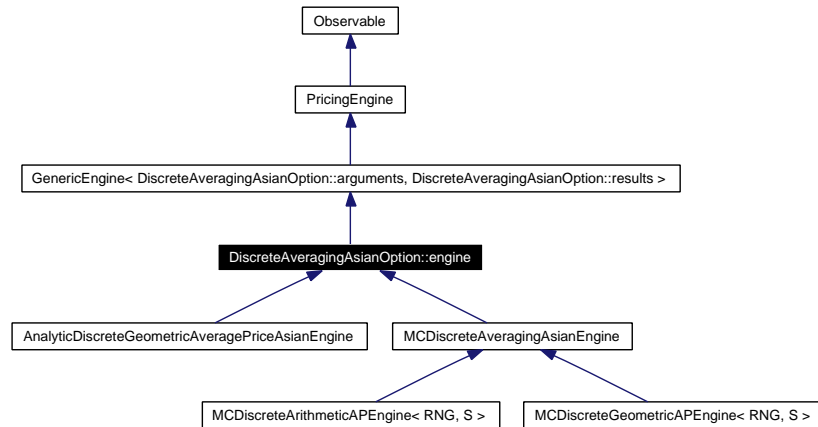
Public Attributes

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

7.163 DiscreteAveragingAsianOption::engine Class Reference

#include <ql/Instruments/asianoption.hpp>

Inheritance diagram for DiscreteAveragingAsianOption::engine:



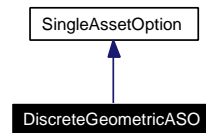
7.163.1 Detailed Description

Discrete-averaging Asian engine base class.

7.164 DiscreteGeometricASO Class Reference

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

Inheritance diagram for DiscreteGeometricASO:



7.164.1 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

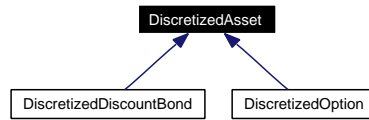
Public Member Functions

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

7.165 DiscretizedAsset Class Reference

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

Inheritance diagram for DiscretizedAsset:



7.165.1 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_`

7.165.2 Member Function Documentation

7.165.2.1 `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](#).

7.165.2.2 `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.

7.165.2.3 `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.

7.165.2.4 `void adjustValues ()`

This method performs both pre- and post-adjustment

7.165.2.5 `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](#).

7.165.2.6 `bool isOnTime (Time t) const` [protected]

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

7.165.2.7 virtual void preAdjustValuesImpl () [protected, virtual]

This method performs the actual pre-adjustment

7.165.2.8 virtual void postAdjustValuesImpl () [protected, virtual]

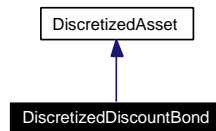
This method performs the actual post-adjustment

Reimplemented in [DiscretizedOption](#).

7.166 DiscretizedDiscountBond Class Reference

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

Inheritance diagram for DiscretizedDiscountBond:



7.166.1 Detailed Description

Useful discretized discount bond asset.

Public Member Functions

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

7.166.2 Member Function Documentation

7.166.2.1 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](#).

7.166.2.2 `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:

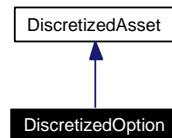
The returned values are not guaranteed to be sorted.

Implements [DiscretizedAsset](#).

7.167 DiscretizedOption Class Reference

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

Inheritance diagram for DiscretizedOption:



7.167.1 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_](#)

7.167.2 Member Function Documentation

7.167.2.1 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](#).

7.167.2.2 `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](#).

7.167.2.3 `void postAdjustValuesImpl ()` [protected, virtual]

This method performs the actual post-adjustment

Reimplemented from [DiscretizedAsset](#).

7.168 Disposable Class Template Reference

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

7.168.1 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;  
}
```

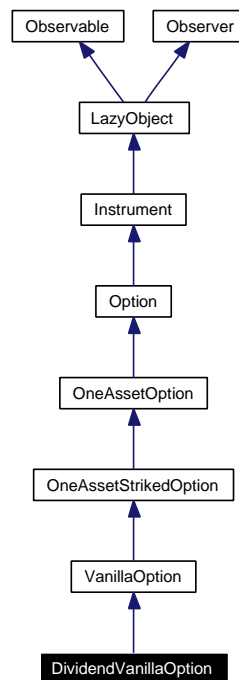
Public Member Functions

- **Disposable** (T &t)
- **Disposable** (const [Disposable](#)< T > &t)
- [Disposable](#)< T > & **operator=** (const [Disposable](#)< T > &t)

7.169 DividendVanillaOption Class Reference

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

Inheritance diagram for DividendVanillaOption:



7.169.1 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](#) > ÷ndDates, const std::vector< [Real](#) > ÷nds, 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.

7.169.2 Member Function Documentation

7.169.2.1 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](#).

7.170 DividendVanillaOption::arguments Class Reference

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

7.170.1 Detailed Description

Arguments for dividend vanilla option calculation

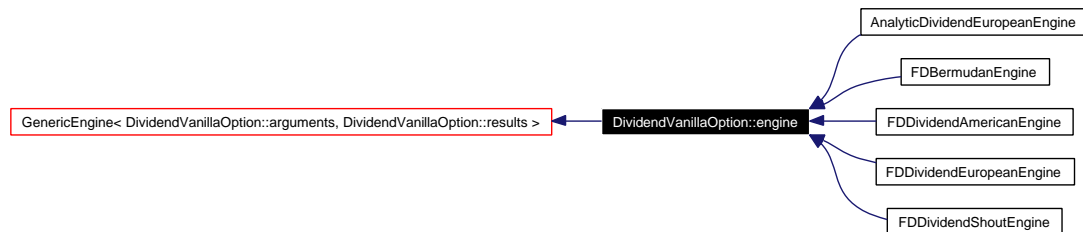
Public Member Functions

- void **validate** () const

7.171 DividendVanillaOption::engine Class Reference

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

Inheritance diagram for DividendVanillaOption::engine:



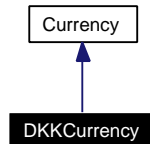
7.171.1 Detailed Description

Dividend vanilla option engine base class.

7.172 DKKCurrency Class Reference

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

Inheritance diagram for DKKCurrency:



7.172.1 Detailed Description

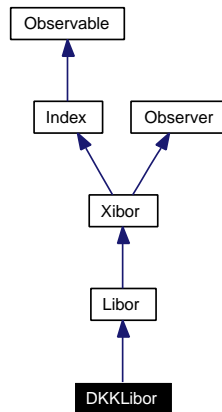
Danish krone.

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

7.173 DKKLibor Class Reference

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

Inheritance diagram for DKKLibor:



7.173.1 Detailed Description

DKK LIBOR rate

Danish Krona LIBOR fixed by BBA.

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

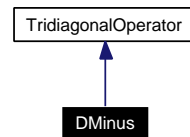
Public Member Functions

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

7.174 DMinus Class Reference

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

Inheritance diagram for DMinus:



7.174.1 Detailed Description

D_- matricial representation

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

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

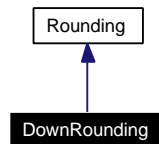
Public Member Functions

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

7.175 DownRounding Class Reference

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

Inheritance diagram for DownRounding:



7.175.1 Detailed Description

Down-rounding.

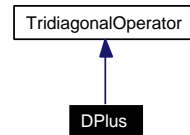
Public Member Functions

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

7.176 DPlus Class Reference

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

Inheritance diagram for DPlus:



7.176.1 Detailed Description

D_+ matricial representation

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

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

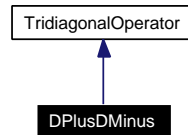
Public Member Functions

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

7.177 DPlusDMinus Class Reference

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

Inheritance diagram for DPlusDMinus:



7.177.1 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.

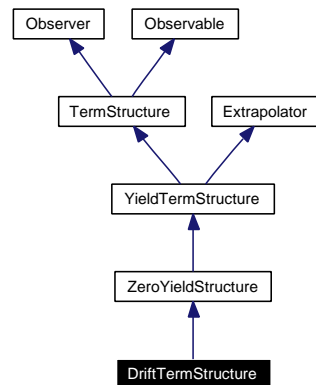
Public Member Functions

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

7.178 DriftTermStructure Class Reference

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

Inheritance diagram for DriftTermStructure:



7.178.1 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](#) > ÷ndTS, 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

7.179 Duration Struct Reference

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

7.179.1 Detailed Description

duration type

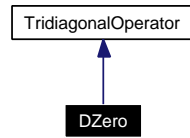
Public Types

- enum Type { Simple, Macaulay, Modified }

7.180 DZero Class Reference

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

Inheritance diagram for DZero:



7.180.1 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.

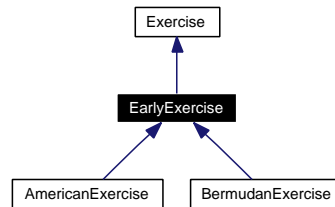
Public Member Functions

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

7.181 EarlyExercise Class Reference

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

Inheritance diagram for EarlyExercise:



7.181.1 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)

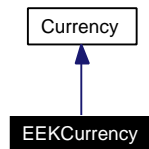
Public Member Functions

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

7.182 EEKCurrency Class Reference

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

Inheritance diagram for EEKCurrency:



7.182.1 Detailed Description

Estonian kroon.

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

7.183 EndCriteria Class Reference

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

7.183.1 Detailed Description

Criteria to end optimization process.

- stationary point
 - stationary gradient
 - maximum number of iterations

Examples:

[BermudanSwaption.cpp](#).

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_`

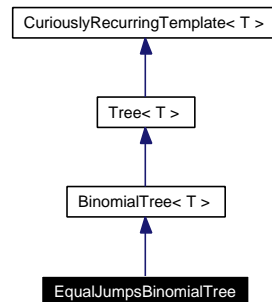
Maximum number of iterations in stationary state.

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

7.184 EqualJumpsBinomialTree Class Template Reference

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

Inheritance diagram for EqualJumpsBinomialTree:



7.184.1 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

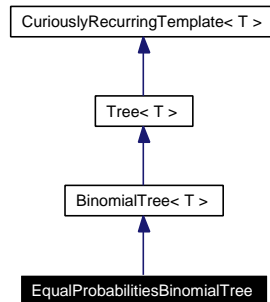
Protected Attributes

- `Real dx_`
- `Real pu_`
- `Real pd_`

7.185 EqualProbabilitiesBinomialTree Class Template Reference

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

Inheritance diagram for EqualProbabilitiesBinomialTree:



7.185.1 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_

7.186 Error Class Reference

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

7.186.1 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.

7.186.2 Constructor & Destructor Documentation

7.186.2.1 [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.

7.186.2.2 [~Error](#) () throw ()

the automatically generated destructor would not have the throw specifier.

7.187 ErrorFunction Class Reference

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

7.187.1 Detailed Description

Error function

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

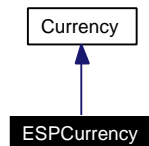
Public Member Functions

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

7.188 ESPCurrency Class Reference

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

Inheritance diagram for ESPCurrency:



7.188.1 Detailed Description

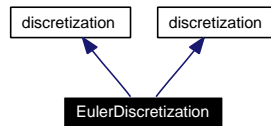
Spanish peseta.

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

7.189 EulerDiscretization Class Reference

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

Inheritance diagram for EulerDiscretization:



7.189.1 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

7.189.2 Member Function Documentation

7.189.2.1 [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](#).

7.189.2.2 [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](#).

7.189.2.3 [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](#).

7.189.2.4 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](#).

7.189.2.5 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](#).

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

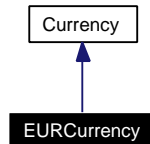
Returns an approximation of the variance defined as $\sigma(t_0, x_0)^2 \Delta t$.

Implements [StochasticProcess1D::discretization](#).

7.190 EURCurrency Class Reference

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

Inheritance diagram for EURCurrency:



7.190.1 Detailed Description

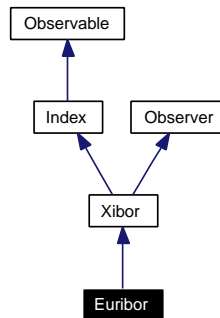
European Euro.

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

7.191 Euribor Class Reference

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

Inheritance diagram for Euribor:



7.191.1 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](#).

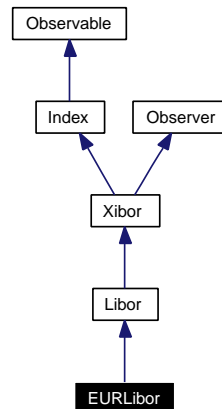
Public Member Functions

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

7.192 EURLibor Class Reference

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

Inheritance diagram for EURLibor:



7.192.1 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.

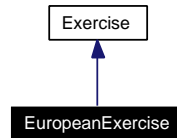
Public Member Functions

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

7.193 EuropeanExercise Class Reference

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

Inheritance diagram for EuropeanExercise:



7.193.1 Detailed Description

European exercise.

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

Examples:

[AmericanOption.cpp](#), and [EuropeanOption.cpp](#).

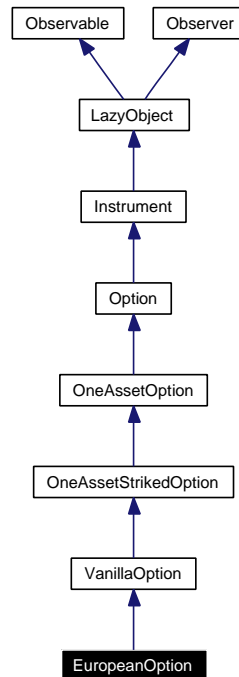
Public Member Functions

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

7.194 EuropeanOption Class Reference

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

Inheritance diagram for EuropeanOption:



7.194.1 Detailed Description

European option on a single asset.

Examples:

[EuropeanOption.cpp](#).

Public Member Functions

- **EuropeanOption** (const boost::shared_ptr< [StochasticProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &, const boost::shared_ptr< [Exercise](#) > &, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())

7.195 ExchangeRate Class Reference

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

7.195.1 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)

7.195.2 Member Enumeration Documentation

7.195.2.1 enum [Type](#)

Enumerator:

Direct given directly by the user

Derived derived from exchange rates between other currencies

7.195.3 Constructor & Destructor Documentation

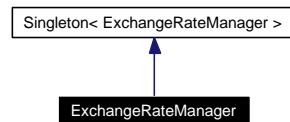
7.195.3.1 [ExchangeRate](#) (const [Currency](#) & *source*, const [Currency](#) & *target*, [Decimal](#) *rate*)

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

7.196 ExchangeRateManager Class Reference

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

Inheritance diagram for ExchangeRateManager:



7.196.1 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](#) >

7.196.2 Member Function Documentation

7.196.2.1 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.

7.196.2.2 `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.

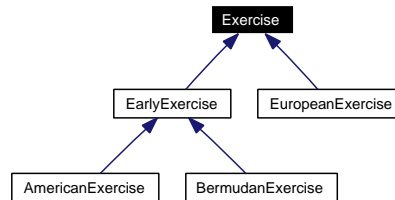
Warning:

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

7.197 Exercise Class Reference

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

Inheritance diagram for Exercise:



7.197.1 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

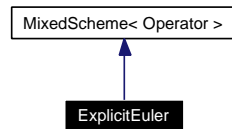
Protected Attributes

- std::vector< **Date** > **dates_**
- Type **type_**

7.198 ExplicitEuler Class Template Reference

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

Inheritance diagram for ExplicitEuler:



7.198.1 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`

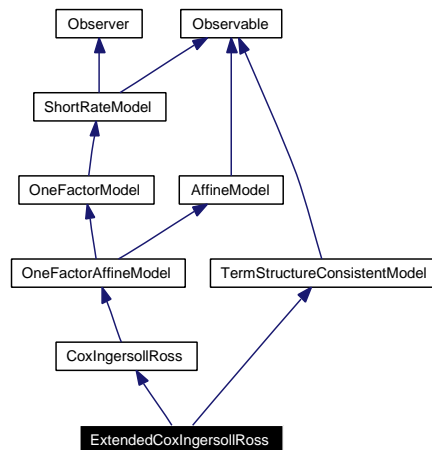
Public Member Functions

- **ExplicitEuler** (const operator_type &L, const std::vector< boost::shared_ptr< bc_type > > &bcs)

7.199 ExtendedCoxIngersollRoss Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss:



7.199.1 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

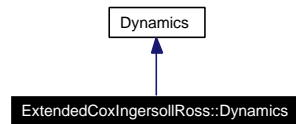
Classes

- class [Dynamics](#)
Short-rate dynamics in the extended Cox-Ingersoll-Ross model.
- class [FittingParameter](#)
Analytical term-structure fitting parameter $\varphi(t)$.

7.200 ExtendedCoxIngersollRoss::Dynamics Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss::Dynamics:



7.200.1 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.

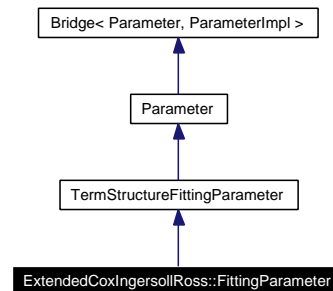
Public Member Functions

- **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

7.201 ExtendedCoxIngersollRoss::FittingParameter Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss::FittingParameter:



7.201.1 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 $h = \sqrt{k^2 + 2\sigma^2}$.

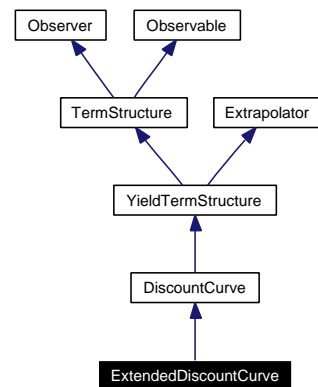
Public Member Functions

- **FittingParameter** (const [Handle< YieldTermStructure >](#) &termStructure, [Real](#) theta, [Real](#) k, [Real](#) sigma, [Real](#) x0)

7.202 ExtendedDiscountCurve Class Reference

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

Inheritance diagram for ExtendedDiscountCurve:



7.202.1 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

7.202.2 Member Function Documentation

7.202.2.1 `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](#).

7.202.2.2 `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.

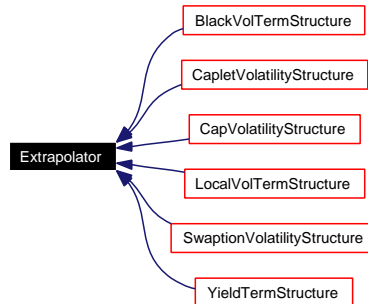
7.202.2.3 `Rate zeroYieldImpl (Time) const` [protected]

Returns the zero yield rate for the given date calculating it from the discount.

7.203 Extrapolator Class Reference

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

Inheritance diagram for Extrapolator:



7.203.1 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

7.204 Factorial Class Reference

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

7.204.1 Detailed Description

Factorial numbers calculator

Tests

the correctness of the returned value is tested by checking it against numerical calculations.

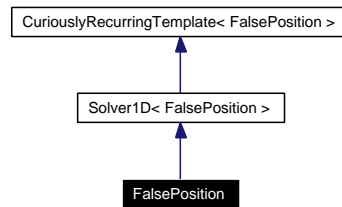
Static Public Member Functions

- static [Real](#) [get](#) ([Natural](#) n)
- static [Real](#) [ln](#) ([Natural](#) n)

7.205 FalsePosition Class Reference

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

Inheritance diagram for FalsePosition:



7.205.1 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`

7.206 FaureRsg Class Reference

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

7.206.1 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**

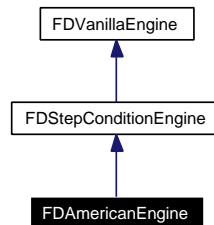
Public Member Functions

- **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

7.207 FDAmericanEngine Class Reference

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

Inheritance diagram for FDAmericanEngine:



7.207.1 Detailed Description

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:

[AmericanOption.cpp](#).

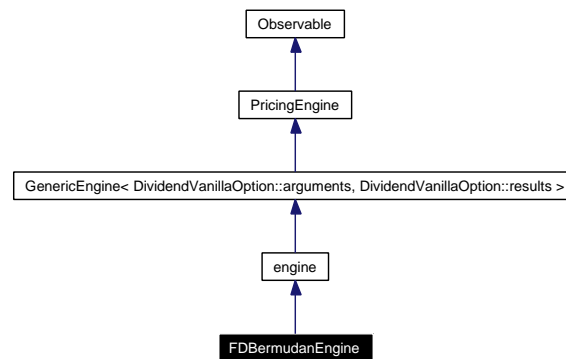
Public Member Functions

- **FDAmericanEngine** ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool timeDependent=false)

7.208 FDBermudanEngine Class Reference

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

Inheritance diagram for FDBermudanEngine:



7.208.1 Detailed Description

Finite-differences Bermudan engine.

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

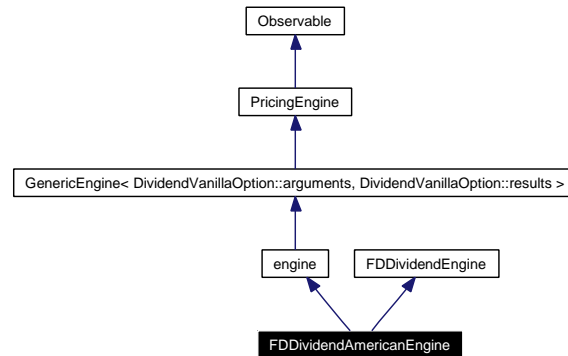
Protected Attributes

- [Real](#) extraTermInBermudan

7.209 FDDividendAmericanEngine Class Reference

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

Inheritance diagram for FDDividendAmericanEngine:



7.209.1 Detailed Description

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

method impliedVolatility() utterly fails

Public Member Functions

- **FDDividendAmericanEngine** ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool time-Dependent=false)
- void **calculate** () const

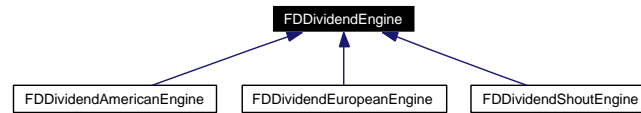
Protected Member Functions

- void **initializeStepCondition** () const

7.210 FDDividendEngine Class Reference

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

Inheritance diagram for FDDividendEngine:



7.210.1 Detailed Description

Base finite-differences pricing engine for dividend options.

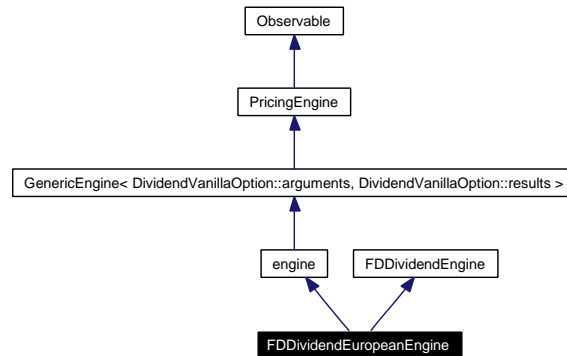
Public Member Functions

- **FDDividendEngine** ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool timeDependent=false)

7.211 FDDividendEuropeanEngine Class Reference

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

Inheritance diagram for FDDividendEuropeanEngine:



7.211.1 Detailed Description

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.

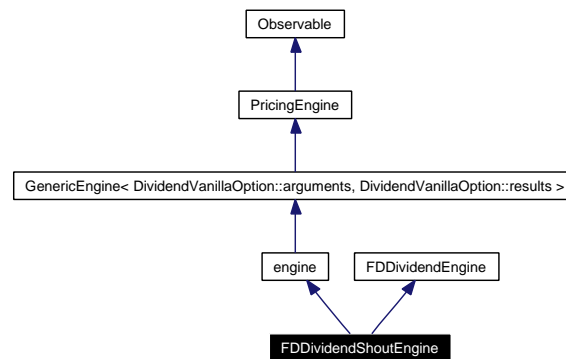
Public Member Functions

- **FDDividendEuropeanEngine** ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool time-Dependent=false)
- void **calculate** () const

7.212 FDDividendShoutEngine Class Reference

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

Inheritance diagram for FDDividendShoutEngine:



7.212.1 Detailed Description

Finite-differences shout engine with dividends.

Public Member Functions

- **FDDividendShoutEngine** ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool timeDependent=false)
- void **calculate** () const

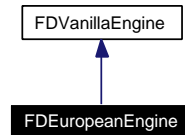
Protected Member Functions

- void **initializeStepCondition** () const

7.213 FDEuropeanEngine Class Reference

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

Inheritance diagram for FDEuropeanEngine:



7.213.1 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:

[EuropeanOption.cpp](#).

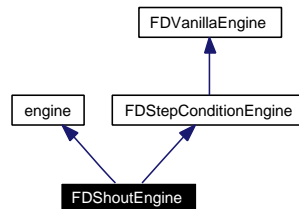
Public Member Functions

- **FDEuropeanEngine** ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool timeDependent=false)

7.214 FDShoutEngine Class Reference

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

Inheritance diagram for FDShoutEngine:



7.214.1 Detailed Description

Finite-differences pricing engine for shout vanilla options.

Tests

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

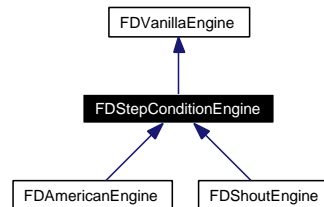
Public Member Functions

- FDShoutEngine ([Size](#) timeSteps=100, [Size](#) gridPoints=100, bool timeDependent=false)

7.215 FDStepConditionEngine Class Reference

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

Inheritance diagram for FDStepConditionEngine:



7.215.1 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
- void **calculate** ([OneAssetOption::results](#) *result) const

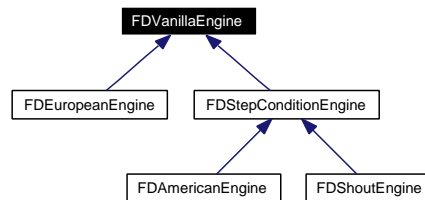
Protected Attributes

- boost::shared_ptr< [StandardStepCondition](#) > **stepCondition_**
- [Array](#) **prices_**
- [TridiagonalOperator](#) **controlOperator_**
- std::vector< boost::shared_ptr< bc_type > > **controlBCs_**
- [Array](#) **controlPrices_**

7.216 FDVanillaEngine Class Reference

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

Inheritance diagram for FDVanillaEngine:



7.216.1 Detailed Description

Finite-differences pricing engine for BSM one asset options.

The name is a misnomer as this is a base class for any finite difference scheme. It's main job is to handle grid layout.

Public Member Functions

- **FDVanillaEngine** ([Size](#) timeSteps, [Size](#) gridPoints, bool timeDependent=false)
- const [Array](#) & grid () const

Protected Types

- typedef [BoundaryCondition](#)< [TridiagonalOperator](#) > **bc_type**

Protected Member Functions

- virtual void **setupArguments** (const [OneAssetOption::arguments](#) *args) const
- virtual void **setGridLimits** () const
- virtual void **setGridLimits** ([Real](#), [Time](#)) const
- virtual void **initializeGrid** () const
- virtual void **initializeInitialCondition** () const
- virtual void **initializeOperator** () const
- virtual [Time](#) **getResidualTime** () const

Protected Attributes

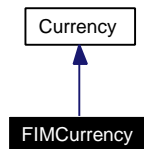
- [Size](#) timeSteps_
- [Size](#) gridPoints_
- bool timeDependent_
- boost::shared_ptr< [BlackScholesProcess](#) > process_
- [Real](#) requiredGridValue_
- [Date](#) exerciseDate_
- [Array](#) grid_

- `boost::shared_ptr< Payoff > payoff_`
- `TridiagonalOperator finiteDifferenceOperator_`
- `Array intrinsicValues_`
- `std::vector< boost::shared_ptr< bc_type > > BCs_`
- `Real sMin_`
- `Real center_`
- `Real sMax_`

7.217 FIMCurrency Class Reference

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

Inheritance diagram for FIMCurrency:



7.217.1 Detailed Description

Finnish markka.

The ISO three-letter code is FIM; the numeric code is 246. It is divided in 100 penniä.

7.218 FiniteDifferenceModel Class Template Reference

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

7.218.1 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)

7.218.2 Member Function Documentation

7.218.2.1 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.

7.218.2.2 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.

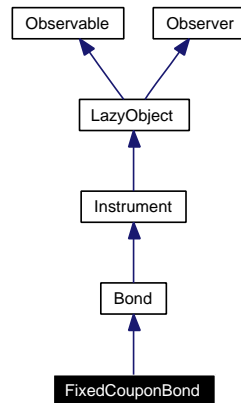
Warning:

being this a rollback, from must be a later time than to.

7.219 FixedCouponBond Class Reference

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

Inheritance diagram for FixedCouponBond:



7.219.1 Detailed Description

fixed-coupon bond

Tests

calculations are tested by checking results against cached values.

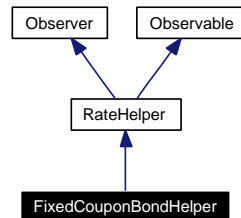
Public Member Functions

- **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)

7.220 FixedCouponBondHelper Class Reference

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

Inheritance diagram for FixedCouponBondHelper:



7.220.1 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)
- **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](#) businessDayConvention_
- [Real](#) redemption_
- [Date](#) stub_

- `bool fromEnd_`
- `Date settlement_`
- `Date latestDate_`
- `boost::shared_ptr< FixedCouponBond > bond_`
- `Handle< YieldTermStructure > termStructureHandle_`

7.220.2 Member Function Documentation

7.220.2.1 `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](#).

7.220.2.2 `void setTermStructure (YieldTermStructure *)` [virtual]

sets the term structure to be used for pricing

Warning:

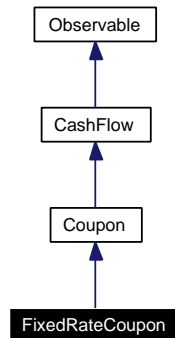
Being a pointer and not a `shared_ptr`, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

7.221 FixedRateCoupon Class Reference

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

Inheritance diagram for FixedRateCoupon:



7.221.1 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](#) &)

7.221.2 Member Function Documentation

7.221.2.1 [Real](#) amount () const [virtual]

returns the amount of the cash flow

Note:

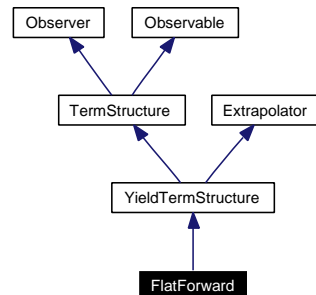
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

7.222 FlatForward Class Reference

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

Inheritance diagram for FlatForward:



7.222.1 Detailed Description

Flat interest-rate curve.

Examples:

[AmericanOption.cpp](#), [BermudanSwaption.cpp](#), [DiscreteHedging.cpp](#), and [EuropeanOption.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** ()

7.222.2 Member Function Documentation

7.222.2.1 void update () [virtual]

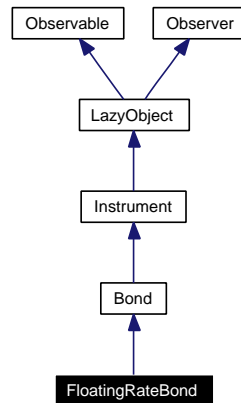
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [TermStructure](#).

7.223 FloatingRateBond Class Reference

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

Inheritance diagram for FloatingRateBond:



7.223.1 Detailed Description

floating-rate bond

Tests

calculations are tested by checking results against cached values.

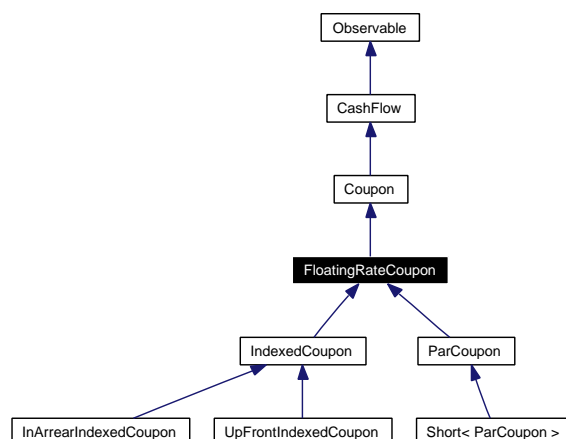
Public Member Functions

- **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)

7.224 FloatingRateCoupon Class Reference

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

Inheritance diagram for FloatingRateCoupon:



7.224.1 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

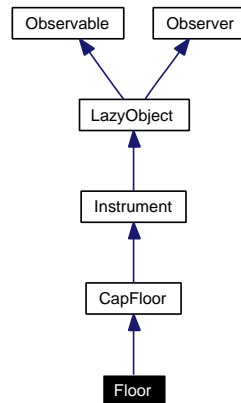
Protected Attributes

- [Integer](#) [fixingDays_](#)
- [Spread](#) [spread_](#)

7.225 Floor Class Reference

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

Inheritance diagram for Floor:



7.225.1 Detailed Description

Concrete floor class.

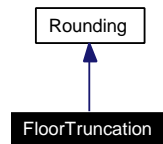
Public Member Functions

- **Floor** (const std::vector< boost::shared_ptr< [CashFlow](#) > > &floatingLeg, const std::vector< [Rate](#) > &exerciseRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared_ptr< [PricingEngine](#) > &engine)

7.226 FloorTruncation Class Reference

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

Inheritance diagram for FloorTruncation:



7.226.1 Detailed Description

[Floor](#) truncation.

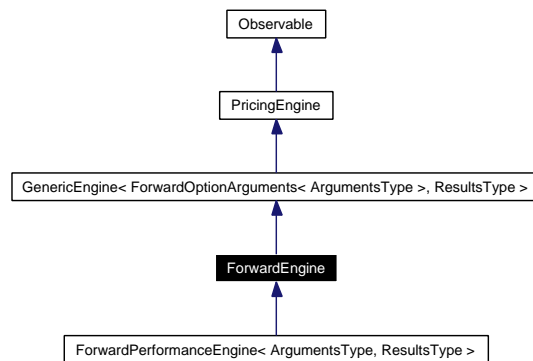
Public Member Functions

- `FloorTruncation` ([Integer](#) precision, [Integer](#) digit=5)

7.227 ForwardEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

Inheritance diagram for ForwardEngine:



7.227.1 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_**

7.228 ForwardFlat Class Reference

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

7.228.1 Detailed Description

Forward-flat interpolation factory and traits.

Public Types

- enum { **global** = 0 }

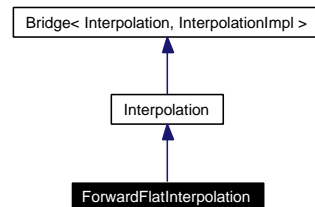
Public Member Functions

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

7.229 ForwardFlatInterpolation Class Reference

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

Inheritance diagram for ForwardFlatInterpolation:



7.229.1 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)`

7.229.2 Constructor & Destructor Documentation

7.229.2.1 [ForwardFlatInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:

the x values must be sorted.

7.230 ForwardOptionArguments Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

7.230.1 Detailed Description

```
template<class ArgumentsType> class QuantLib::ForwardOptionArguments< Arguments-  
Type >
```

Arguments for forward (strike-resetting) option calculation

Public Member Functions

- void `validate ()` const

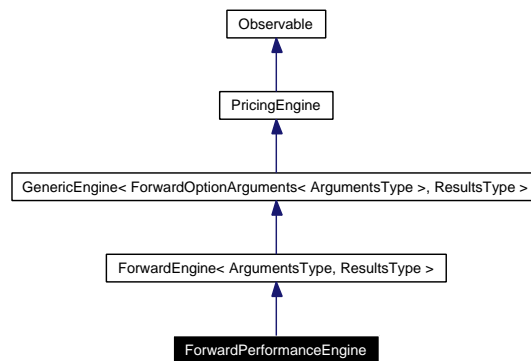
Public Attributes

- [Real](#) `moneyness`
- [Date](#) `resetDate`

7.231 ForwardPerformanceEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardperformanceengine.hpp>
```

Inheritance diagram for ForwardPerformanceEngine:



7.231.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::ForwardPerformanceEngine< 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

7.232 ForwardRate Struct Reference

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

7.232.1 Detailed Description

Forward-curve traits.

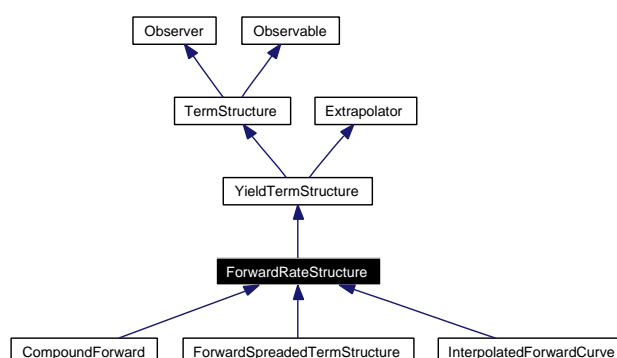
Static Public Member Functions

- 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)

7.233 ForwardRateStructure Class Reference

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

Inheritance diagram for ForwardRateStructure:



7.233.1 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

7.233.2 Member Function Documentation

7.233.2.1 [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](#).

7.233.2.2 [Rate](#) `zeroYieldImpl (Time) const` [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

Warning:

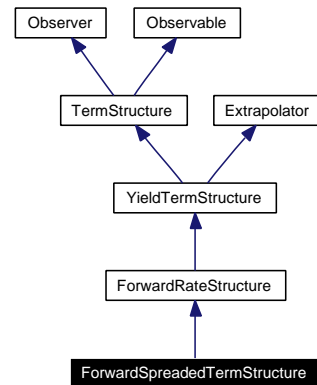
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own `zeroYield` method.

Reimplemented in [CompoundForward](#), [InterpolatedForwardCurve](#), and [ForwardSpreadedTermStructure](#).

7.234 ForwardSpreadedTermStructure Class Reference

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

Inheritance diagram for ForwardSpreadedTermStructure:



7.234.1 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

7.234.2 Member Function Documentation

7.234.2.1 [Rate zeroYieldImpl](#) (Time) const [protected, virtual]

returns the spreaded zero yield rate

Warning:

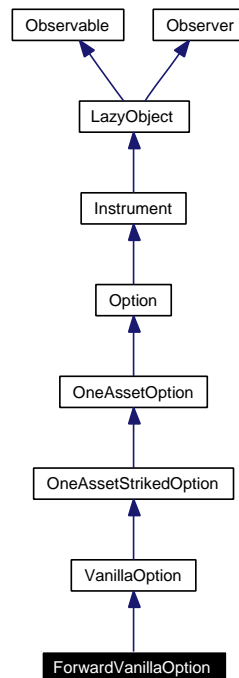
This method must disappear should the spread become a curve

Reimplemented from [ForwardRateStructure](#).

7.235 ForwardVanillaOption Class Reference

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

Inheritance diagram for ForwardVanillaOption:



7.235.1 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

Protected Member Functions

- void [performCalculations](#) () const

7.235.2 Member Function Documentation

7.235.2.1 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](#).

7.235.2.2 void performCalculations () const [protected, virtual]

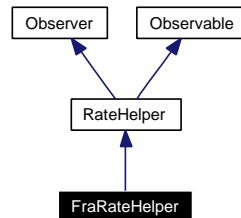
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [OneAssetStrikedOption](#).

7.236 FraRateHelper Class Reference

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

Inheritance diagram for FraRateHelper:



7.236.1 Detailed Description

Forward rate agreement.

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

7.236.2 Member Function Documentation

7.236.2.1 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](#).

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

latest relevant date

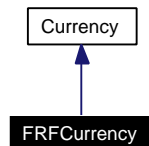
The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

7.237 FRFCurrency Class Reference

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

Inheritance diagram for FRFCurrency:



7.237.1 Detailed Description

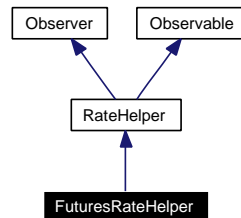
French franc.

The ISO three-letter code is FRF; the numeric code is 250. It is divided in 100 centimes.

7.238 FuturesRateHelper Class Reference

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

Inheritance diagram for FuturesRateHelper:



7.238.1 Detailed Description

Interest-rate futures.

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

7.238.2 Member Function Documentation

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

latest relevant date

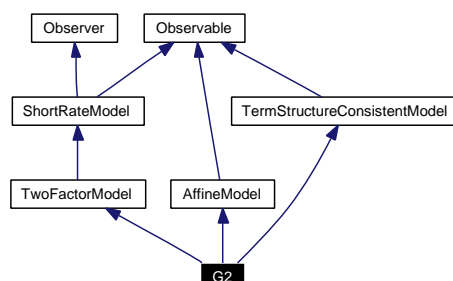
The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

7.239 G2 Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Inheritance diagram for G2:



7.239.1 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.
- [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**

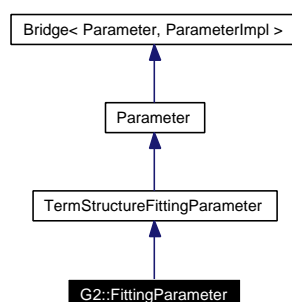
Classes

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

7.240 G2::FittingParameter Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Inheritance diagram for G2::FittingParameter:



7.240.1 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 .

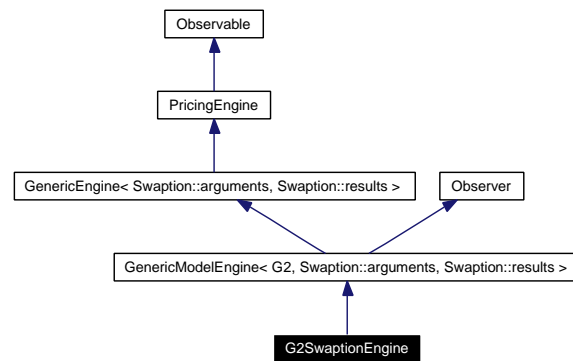
Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a, [Real](#) sigma, [Real](#) b, [Real](#) eta, [Real](#) rho)

7.241 G2SwaptionEngine Class Reference

```
#include <ql/PricingEngines/Swaption/g2swaptionengine.hpp>
```

Inheritance diagram for G2SwaptionEngine:



7.241.1 Detailed Description

Swaption priced by means of the Black formula

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- **G2SwaptionEngine** (const boost::shared_ptr< [G2](#) > &mod, [Real](#) range, [Size](#) intervals)
- void **calculate** () const

7.242 GammaFunction Class Reference

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

7.242.1 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.

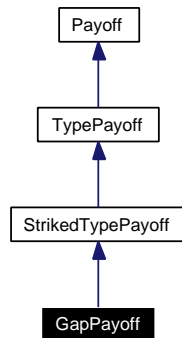
Public Member Functions

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

7.243 GapPayoff Class Reference

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

Inheritance diagram for GapPayoff:



7.243.1 Detailed Description

Binary gap payoff.

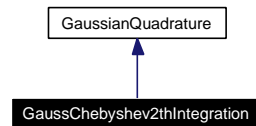
Public Member Functions

- **GapPayoff** (Option::Type type, [Real](#) strike, [Real](#) strikePayoff)
- **Real operator()** ([Real](#) price) const
- **Real strikePayoff** () const

7.244 GaussChebyshev2thIntegration Class Reference

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

Inheritance diagram for GaussChebyshev2thIntegration:



7.244.1 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

$$w(x) = (1 - x^2)^{1/2}$$

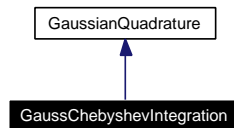
Public Member Functions

- `GaussChebyshev2thIntegration` ([Size n](#))

7.245 GaussChebyshevIntegration Class Reference

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

Inheritance diagram for GaussChebyshevIntegration:



7.245.1 Detailed Description

Gauss-Chebyshev integration.

This class performs a 1-dimensional Gauss-Chebyshev integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{-1/2}$$

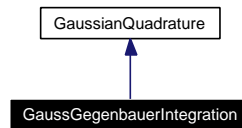
Public Member Functions

- `GaussChebyshevIntegration` ([Size](#) n)

7.246 GaussGegenbauerIntegration Class Reference

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

Inheritance diagram for GaussGegenbauerIntegration:



7.246.1 Detailed Description

Gauss-Gegenbauer integration.

This class performs a 1-dimensional Gauss-Gegenbauer integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = (1 - x^2)^{\lambda-1/2}$$

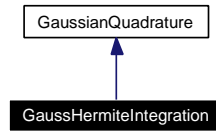
Public Member Functions

- `GaussGegenbauerIntegration` ([Size](#) n, [Real](#) lambda)

7.247 GaussHermiteIntegration Class Reference

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

Inheritance diagram for GaussHermiteIntegration:



7.247.1 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

$$\mu > -0.5$$

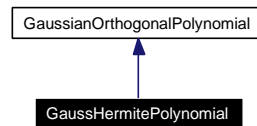
Public Member Functions

- `GaussHermiteIntegration` ([Size](#) n, [Real](#) mu=0.0)

7.248 GaussHermitePolynomial Class Reference

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

Inheritance diagram for GaussHermitePolynomial:



7.248.1 Detailed Description

Gauss-Hermite polynomial.

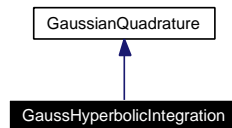
Public Member Functions

- `GaussHermitePolynomial` ([Real](#) mu=0.0)
- [Real](#) `mu_0` () const
- [Real](#) `alpha` ([Size](#) i) const
- [Real](#) `beta` ([Size](#) i) const
- [Real](#) `w` ([Real](#) x) const

7.249 GaussHyperbolicIntegration Class Reference

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

Inheritance diagram for GaussHyperbolicIntegration:



7.249.1 Detailed Description

Gauss-Hyperbolic integration.

This class performs a 1-dimensional Gauss-Hyperbolic integration.

$$\int_{-\infty}^{\infty} f(x) dx$$

The weighting function is

$$w(x) = 1/\cosh(x)$$

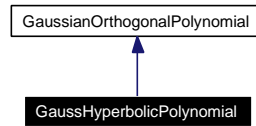
Public Member Functions

- `GaussHyperbolicIntegration` ([Size](#) n)

7.250 GaussHyperbolicPolynomial Class Reference

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

Inheritance diagram for GaussHyperbolicPolynomial:



7.250.1 Detailed Description

Gauss hyperbolic polynomial.

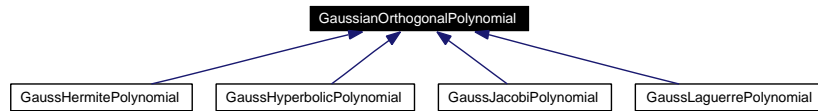
Public Member Functions

- [Real mu_0](#) () const
- [Real alpha](#) ([Size i](#)) const
- [Real beta](#) ([Size i](#)) const
- [Real w](#) ([Real x](#)) const

7.251 GaussianOrthogonalPolynomial Class Reference

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

Inheritance diagram for GaussianOrthogonalPolynomial:



7.251.1 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

$$\mu_0 = \int w(x)dx$$

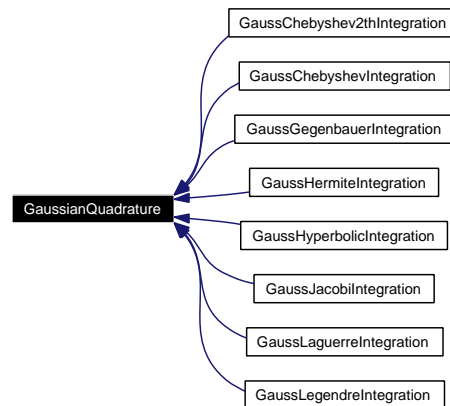
Public Member Functions

- 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

7.252 GaussianQuadrature Class Reference

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

Inheritance diagram for GaussianQuadrature:



7.252.1 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

7.253 GaussianStatistics Class Template Reference

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

7.253.1 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)

7.253.2 Member Function Documentation

7.253.2.1 [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 $x < 0$

7.253.2.2 Real gaussianDownsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.253.2.3 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)

7.253.2.4 Real gaussianPercentile (Real percentile) const

gaussian-assumption y-th percentile, defined as the value x such that

$$y = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp(-u^2/2) du$$

7.253.2.5 Real gaussianTopPercentile (Real percentile) const

Precondition:

percentile must be in range (0-100%) extremes excluded

7.253.2.6 Real gaussianPotentialUpside (Real percentile) const

gaussian-assumption Potential-Upside at a given percentile

Precondition:

percentile must be in range [90-100%)

7.253.2.7 Real gaussianValueAtRisk (Real percentile) const

gaussian-assumption Value-At-Risk at a given percentile

Precondition:

percentile must be in range [90-100%)

7.253.2.8 Real gaussianExpectedShortfall (Real percentile) const

gaussian-assumption Expected Shortfall at a given percentile

Assuming a gaussian distribution it returns the expected loss in case that the loss exceeded a VaR threshold,

$$E[x \mid x < \text{VaR}(p)],$$

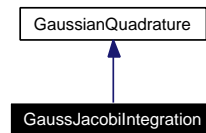
that is the average of observations below the given percentile p . Also know as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

7.254 GaussJacobiIntegration Class Reference

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

Inheritance diagram for GaussJacobiIntegration:



7.254.1 Detailed Description

Gauss-Jacobi integration.

This class performs a 1-dimensional Gauss-Jacobi integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x; \alpha, \beta) = (1 - x)^\alpha (1 + x)^\beta$$

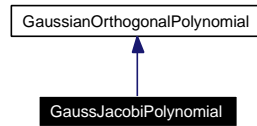
Public Member Functions

- `GaussJacobiIntegration` ([Size](#) n, [Real](#) alpha, [Real](#) beta)

7.255 GaussJacobiPolynomial Class Reference

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

Inheritance diagram for GaussJacobiPolynomial:



7.255.1 Detailed Description

Gauss-Jacobi polynomial.

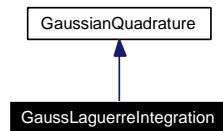
Public Member Functions

- `GaussJacobiPolynomial` ([Real](#) alpha, [Real](#) beta)
- [Real](#) `mu_0` () const
- [Real](#) `alpha` ([Size](#) i) const
- [Real](#) `beta` ([Size](#) i) const
- [Real](#) `w` ([Real](#) x) const

7.256 GaussLaguerreIntegration Class Reference

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

Inheritance diagram for GaussLaguerreIntegration:



7.256.1 Detailed Description

generalized Gauss-Laguerre integration

This class performs a 1-dimensional Gauss-Laguerre integration.

$$\int_0^{\text{inf}} f(x) dx$$

The weighting function is

$$w(x; s) = x^s \exp -x$$

and

$$s > -1$$

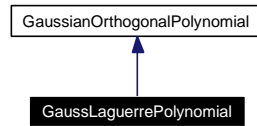
Public Member Functions

- `GaussLaguerreIntegration` ([Size](#) n, [Real](#) s=0.0)

7.257 GaussLaguerrePolynomial Class Reference

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

Inheritance diagram for GaussLaguerrePolynomial:



7.257.1 Detailed Description

Gauss-Laguerre polynomial.

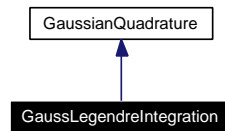
Public Member Functions

- `GaussLaguerrePolynomial` (`Real` s=0.0)
- `Real` `mu_0` () const
- `Real` `alpha` (`Size` i) const
- `Real` `beta` (`Size` i) const
- `Real` `w` (`Real` x) const

7.258 GaussLegendreIntegration Class Reference

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

Inheritance diagram for GaussLegendreIntegration:



7.258.1 Detailed Description

Gauss-Legendre integration.

This class performs a 1-dimensional Gauss-Legendre integration.

$$\int_{-1}^1 f(x) dx$$

The weighting function is

$$w(x) = 1$$

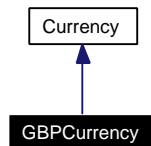
Public Member Functions

- `GaussLegendreIntegration` ([Size](#) n)

7.259 GBPCurrency Class Reference

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

Inheritance diagram for GBPCurrency:



7.259.1 Detailed Description

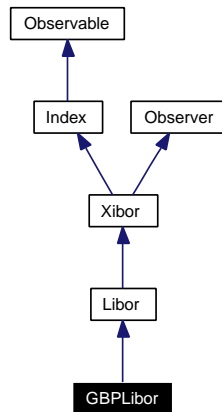
British pound sterling.

The ISO three-letter code is GBP; the numeric code is 826. It is divided into 100 pence.

7.260 GBPLibor Class Reference

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

Inheritance diagram for GBPLibor:



7.260.1 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](#)())

7.261 GeneralStatistics Class Reference

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

7.261.1 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

7.261.2 Member Function Documentation

7.261.2.1 Real mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

7.261.2.2 Real variance () const

returns the variance, defined as

$$\sigma^2 = \frac{N}{N-1} \langle (x - \langle x \rangle)^2 \rangle.$$

7.261.2.3 Real standardDeviation () const

returns the standard deviation σ , defined as the square root of the variance.

7.261.2.4 Real errorEstimate () const

returns the error estimate on the mean value, defined as $\epsilon = \sigma / \sqrt{N}$.

7.261.2.5 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.

7.261.2.6 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.

7.261.2.7 Real min () const

returns the minimum sample value

7.261.2.8 Real max () const

returns the maximum sample value

7.261.2.9 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.

7.261.2.10 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 $(0 - 1]$.

7.261.2.11 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 $(0 - 1]$.

7.261.2.12 void add (Real value, Real weight = 1.0)

adds a datum to the set, possibly with a weight

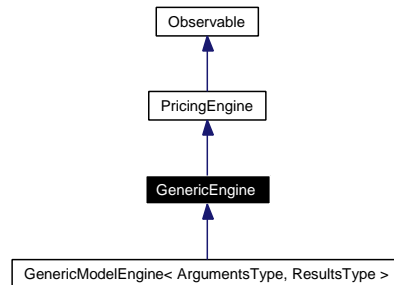
Precondition:

weights must be positive or null

7.262 GenericEngine Class Template Reference

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

Inheritance diagram for GenericEngine:



7.262.1 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`

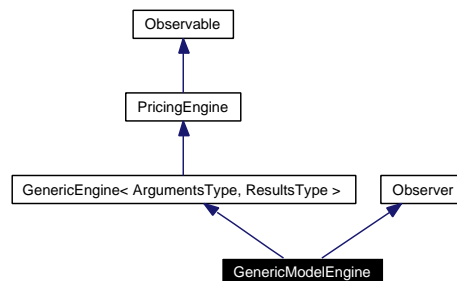
Protected Attributes

- `ArgumentsType arguments_`
- `ResultsType results_`

7.263 GenericModelEngine Class Template Reference

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

Inheritance diagram for GenericModelEngine:



7.263.1 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_**

7.263.2 Member Function Documentation

7.263.2.1 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 >](#), and [LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >](#).

7.264 GenericRiskStatistics Class Template Reference

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

7.264.1 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

7.264.2 Member Function Documentation

7.264.2.1 [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).

7.264.2.2 [Real semiDeviation](#) () const

returns the semi deviation, defined as the square root of the semi variance.

7.264.2.3 Real downsideVariance () const

returns the variance of observations below 0.0,

$$\frac{N}{N-1} \mathbb{E}[x^2 \mid x < 0].$$

7.264.2.4 Real downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.264.2.5 Real regret (Real target) const

returns the variance of observations below target,

$$\frac{N}{N-1} \mathbb{E}[(x - t)^2 \mid x < t].$$

See Dembo and Freeman, "The Rules Of Risk", Wiley (2001).

7.264.2.6 Real potentialUpside (Real centile) const

potential upside (the reciprocal of VAR) at a given percentile

Precondition:

percentile must be in range [90-100%)

7.264.2.7 Real valueAtRisk (Real centile) const

value-at-risk at a given percentile

Precondition:

percentile must be in range [90-100%)

7.264.2.8 Real expectedShortfall (Real percentile) const

expected shortfall at a given percentile

returns the expected loss in case that the loss exceeded a VaR threshold,

$$\mathbb{E}[x \mid x < \text{VaR}(p)],$$

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)

7.264.2.9 Real shortfall (Real target) const

probability of missing the given target, defined as

$$E[\Theta \mid (-\infty, \infty)]$$

where

$$\Theta(x) = \begin{cases} 1 & x < t \\ 0 & x \geq t \end{cases}$$

7.264.2.10 Real averageShortfall (Real target) const

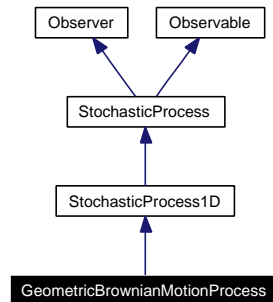
averaged shortfallness, defined as

$$E[t - x \mid x < t]$$

7.265 GeometricBrownianMotionProcess Class Reference

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

Inheritance diagram for GeometricBrownianMotionProcess:



7.265.1 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. $\sigma(t, x_t)$

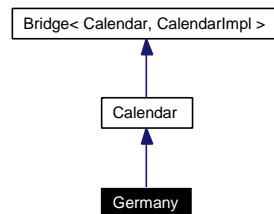
Protected Attributes

- double **initialValue_**
- double **mue_**
- double **sigma_**

7.266 Germany Class Reference

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

Inheritance diagram for Germany:



7.266.1 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](#))

7.266.2 Member Enumeration Documentation

7.266.2.1 enum [Market](#)

German calendars.

Enumerator:

Settlement generic settlement calendar

FrankfurtStockExchange Frankfurt stock-exchange.

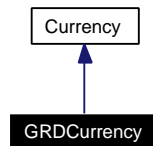
Xetra Xetra.

Eurex Eurex.

7.267 GRDCurrency Class Reference

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

Inheritance diagram for GRDCurrency:



7.267.1 Detailed Description

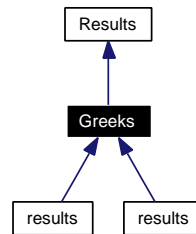
Greek drachma.

The ISO three-letter code is GRD; the numeric code is 300. It is divided in 100 lepta.

7.268 Greeks Class Reference

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

Inheritance diagram for Greeks:



7.268.1 Detailed Description

additional option results

Public Member Functions

- `void reset ()`

Public Attributes

- [Real](#) delta
- [Real](#) gamma
- [Real](#) theta
- [Real](#) vega
- [Real](#) rho
- [Real](#) dividendRho

7.269 HaltonRsg Class Reference

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

7.269.1 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

7.270 Handle Class Template Reference

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

7.270.1 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](#)

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.

7.270.2 Constructor & Destructor Documentation

7.270.2.1 [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.

7.270.3 Member Function Documentation

7.270.3.1 void [linkTo](#) (const boost::shared_ptr< Type > &, bool registerAsObserver = true)

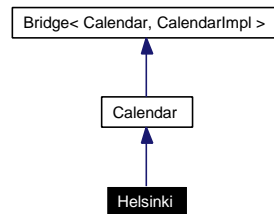
Warning:

see the documentation of the [Link](#) class for issues relatives to registerAsObserver.

7.271 Helsinki Class Reference

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

Inheritance diagram for Helsinki:



7.271.1 Detailed Description

Helsinki calendar

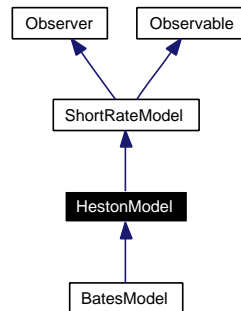
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- Ascension Thursday
- Labour Day, May 1st
- Midsummer Eve (Friday between June 18-24)
- Independence Day, December 6th
- Christmas Eve, December 24th
- Christmas, December 25th
- Boxing Day, December 26th

7.272 HestonModel Class Reference

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

Inheritance diagram for HestonModel:



7.272.1 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.

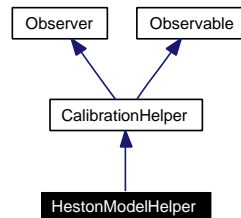
Public Member Functions

- **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

7.273 HestonModelHelper Class Reference

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

Inheritance diagram for HestonModelHelper:



7.273.1 Detailed Description

calibration helper for Heston model

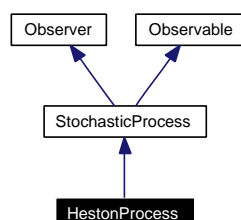
Public Member Functions

- **HestonModelHelper** (const [Period](#) &maturity, const [Calendar](#) &calendar, const [Real](#) s0, const [Real](#) strikePrice, const [Handle](#)< [Quote](#) > &volatility, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [YieldTermStructure](#) > ÷ndYield, 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
- [Real](#) **calibrationError** ()
returns the error resulting from the model valuation
- [Time](#) **maturity** () const

7.274 HestonProcess Class Reference

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

Inheritance diagram for HestonProcess:



7.274.1 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](#) > ÷ndYield, 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

7.274.2 Member Function Documentation

7.274.2.1 [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](#).

7.274.2.2 [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](#).

7.275 History Class Reference

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

7.275.1 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

7.275.2 Constructor & Destructor Documentation

7.275.2.1 [History \(\)](#)

Default constructor

7.275.2.2 [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.

7.275.2.3 **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.

7.275.2.4 **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.

7.276 History::const_iterator Class Reference

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

7.276.1 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**

7.277 History::Entry Class Reference

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

7.277.1 Detailed Description

single datum in history

Public Member Functions

- const [Date](#) & **date** () const
- [Real](#) **value** () const

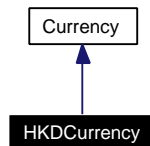
Friends

- class **const_iterator**

7.278 HKDCurrency Class Reference

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

Inheritance diagram for HKDCurrency:



7.278.1 Detailed Description

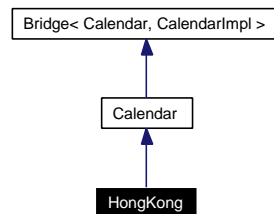
Honk Kong dollar.

The ISO three-letter code is HKD; the numeric code is 344. It is divided in 100 cents.

7.279 HongKong Class Reference

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

Inheritance diagram for HongKong:



7.279.1 Detailed Description

Hong Kong calendar.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Ching Ming Festival, April 5th
- Good Friday
- Easter Monday
- Labor Day, May 1st
- SAR Establishment Day, July 1st
- National Day, October 1st
- Christmas, December 25th
- Boxing Day, December 26th
- Christmas Holiday, December 27th

Other holidays for which no rule is given (data available for 2004/2005 only:)

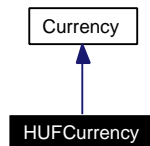
- Lunar New Year
- Chinese New Year
- Buddha's birthday
- Tuen NG Festival
- Mid-autumn fest
- Chung Yeung fest

Data from <http://www.hkex.com.hk>

7.280 HUFCurrency Class Reference

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

Inheritance diagram for HUFCurrency:



7.280.1 Detailed Description

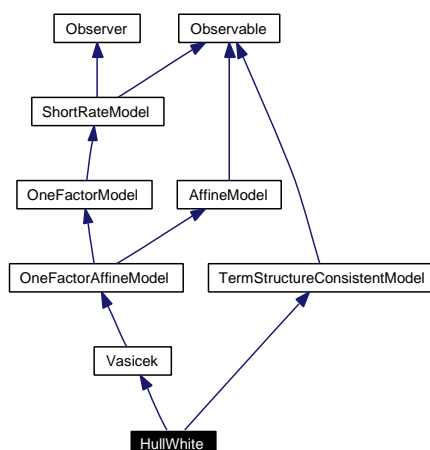
Hungarian forint.

The ISO three-letter code is HUF; the numeric code is 348. It has no subdivisions.

7.281 HullWhite Class Reference

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

Inheritance diagram for HullWhite:



7.281.1 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 α and σ 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 $\varphi(t)$.

7.282 HullWhite::Dynamics Class Reference

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

7.282.1 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.

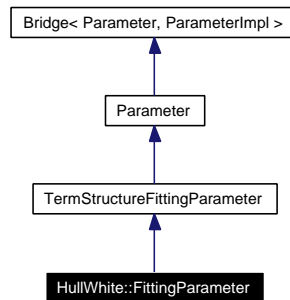
Public Member Functions

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

7.283 HullWhite::FittingParameter Class Reference

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

Inheritance diagram for HullWhite::FittingParameter:



7.283.1 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 .

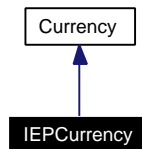
Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a, [Real](#) sigma)

7.284 IEPCurrency Class Reference

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

Inheritance diagram for IEPCurrency:



7.284.1 Detailed Description

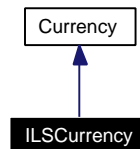
Irish punt.

The ISO three-letter code is IEP; the numeric code is 372. It is divided in 100 pence.

7.285 ILSCurrency Class Reference

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

Inheritance diagram for ILSCurrency:



7.285.1 Detailed Description

Israeli shekel.

The ISO three-letter code is ILS; the numeric code is 376. It is divided in 100 agorot.

7.286 IMM Struct Reference

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

7.286.1 Detailed Description

Main cycle of the International [Money](#) Market (a.k.a. [IMM](#)) Months.

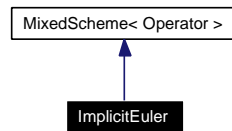
Public Types

- enum `Month` { `H` = 3, `M` = 6, `U` = 9, `Z` = 12 }

7.287 ImplicitEuler Class Template Reference

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

Inheritance diagram for ImplicitEuler:



7.287.1 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**

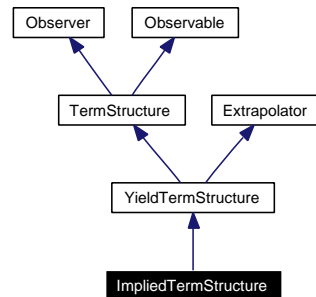
Public Member Functions

- **ImplicitEuler** (const operator_type &L, const bc_set &bcs)

7.288 ImpliedTermStructure Class Reference

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

Inheritance diagram for ImpliedTermStructure:



7.288.1 Detailed Description

Implied term structure at a given date in the future.

The given date will be the implied reference date.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Tests

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure is checked.

Public Member Functions

- **ImpliedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Date](#) &reference-Date)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return rates

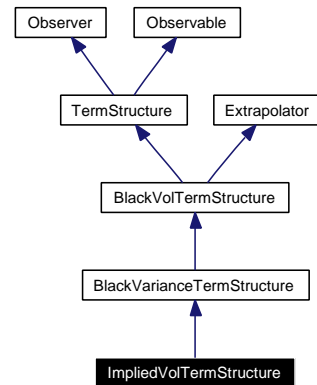
Protected Member Functions

- [DiscountFactor](#) `discountImpl` ([Time](#)) const
returns the discount factor as seen from the evaluation date

7.289 ImpliedVolTermStructure Class Reference

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

Inheritance diagram for ImpliedVolTermStructure:



7.289.1 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](#) &)

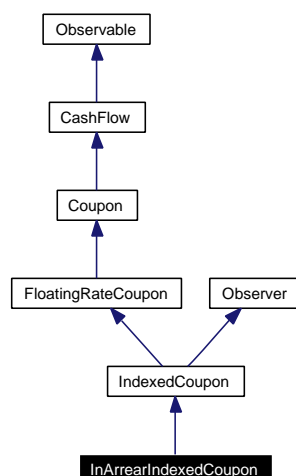
Protected Member Functions

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

7.290 InArrearIndexedCoupon Class Reference

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

Inheritance diagram for InArrearIndexedCoupon:



7.290.1 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

- [RateConvexityAdjustment](#) ([Rate](#) fixing) const
convexity adjustment for the given index fixing

Protected Attributes

- boost::shared_ptr< [Xibor](#) > [xibor_](#)
- [Handle](#)< [CapletVolatilityStructure](#) > [capletVolatility_](#)

7.291 IncrementalStatistics Class Reference

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

7.291.1 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_

7.291.2 Member Function Documentation

7.291.2.1 [Real](#) mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

7.291.2.2 [Real](#) variance () const

returns the variance, defined as

$$\frac{N}{N-1} \langle (x - \langle x \rangle)^2 \rangle.$$

7.291.2.3 [Real](#) standardDeviation () const

returns the standard deviation σ , defined as the square root of the variance.

7.291.2.4 [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 $x < 0$

7.291.2.5 [Real](#) downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.291.2.6 Real errorEstimate () const

returns the error estimate ϵ , defined as the square root of the ratio of the variance to the number of samples.

7.291.2.7 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.

7.291.2.8 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.

7.291.2.9 Real min () const

returns the minimum sample value

7.291.2.10 Real max () const

returns the maximum sample value

7.291.2.11 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

7.291.2.12 void addSequence (DataIterator begin, DataIterator end, WeightIterator wbegin)

adds a sequence of data to the set, each with its weight

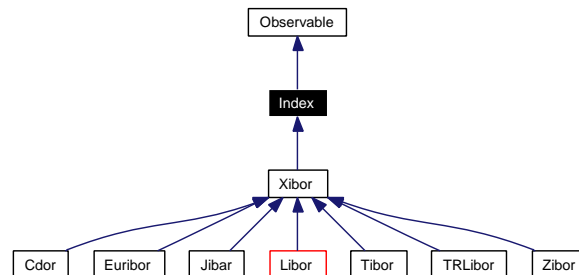
Precondition:

weights must be positive or null

7.292 Index Class Reference

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

Inheritance diagram for Index:



7.292.1 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

7.292.2 Member Function Documentation

7.292.2.1 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](#).

7.292.2.2 virtual `Rate fixing (const Date &fixingDate) const` [pure virtual]

returns the fixing at the given date

Note:

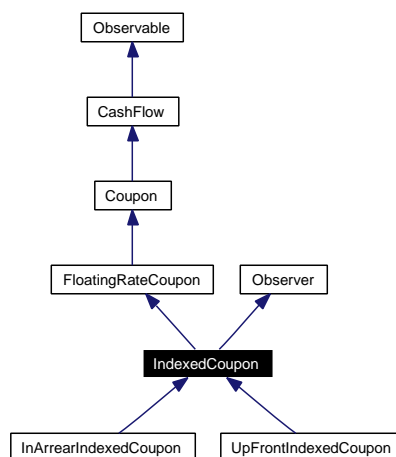
any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implemented in [Xibor](#).

7.293 IndexedCoupon Class Reference

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

Inheritance diagram for IndexedCoupon:



7.293.1 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 &)`

7.293.2 Member Function Documentation

7.293.2.1 [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](#).

7.293.2.2 `void update ()` [virtual]

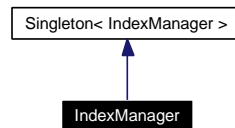
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.294 IndexManager Class Reference

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

Inheritance diagram for IndexManager:



7.294.1 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

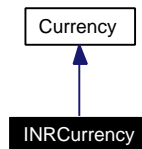
Friends

- class `Singleton< IndexManager >`

7.295 INRCurrency Class Reference

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

Inheritance diagram for INRCurrency:



7.295.1 Detailed Description

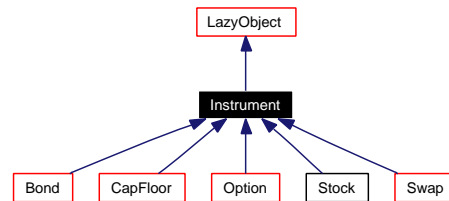
Indian rupee.

The ISO three-letter code is INR; the numeric code is 356. It is divided in 100 paise.

7.296 Instrument Class Reference

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

Inheritance diagram for Instrument:



7.296.1 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

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_`

7.296.2 Member Function Documentation

7.296.2.1 `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.

7.296.2.2 `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](#), [ConvertibleBond::option](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoForwardVanillaOption](#), [QuantoVanillaOption](#), [SimpleSwap](#), and [Swaption](#).

7.296.2.3 `void calculate () const [protected, virtual]`

This method performs all needed calculations by calling the **performCalculations** method.

Warning:

Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.

Should this method be redefined in derived classes, [LazyObject::calculate\(\)](#) should be called in the overriding method.

Reimplemented from [LazyObject](#).

7.296.2.4 `void setupExpired () const [protected, virtual]`

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

Reimplemented in [MultiAssetOption](#), [OneAssetOption](#), [QuantoVanillaOption](#), and [Swap](#).

7.296.2.5 void performCalculations () const [protected, virtual]

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

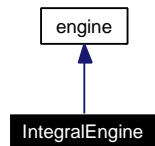
Implements [LazyObject](#).

Reimplemented in [BarrierOption](#), [Bond](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), [Stock](#), and [Swap](#).

7.297 IntegralEngine Class Reference

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

Inheritance diagram for IntegralEngine:



7.297.1 Detailed Description

Pricing engine for European vanilla options using integral approach

Todo

define tolerance for calculate()

Examples:

[EuropeanOption.cpp](#).

Public Member Functions

- void **calculate** () const

7.298 InterestRate Class Reference

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

7.298.1 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 &)`

7.298.2 Member Function Documentation

7.298.2.1 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.

7.298.2.2 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.

7.298.2.3 **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.

7.298.2.4 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.

7.298.2.5 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.

7.298.2.6 **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.

7.298.2.7 **InterestRate** equivalentRate (**Date** d1, **Date** d2, const **DayCounter** & resultDC, Compounding comp, **Frequency** freq = Annual) const

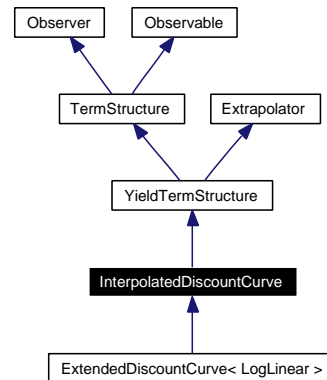
equivalent rate for a compounding period between two dates

The resulting rate is calculated taking the required day-counting rule into account.

7.299 InterpolatedDiscountCurve Class Template Reference

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

Inheritance diagram for InterpolatedDiscountCurve:



7.299.1 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

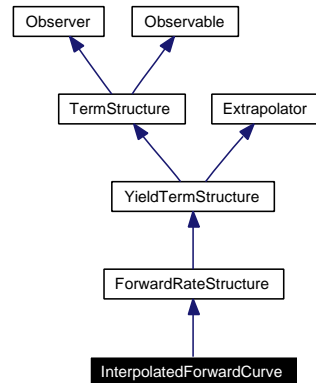
Protected Attributes

- [DayCounter](#) **dayCounter_**
- std::vector< [Date](#) > **dates_**
- std::vector< [Time](#) > **times_**
- std::vector< [DiscountFactor](#) > **data_**
- [Interpolation](#) **interpolation_**
- Interpolator **interpolator_**

7.300 InterpolatedForwardCurve Class Template Reference

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

Inheritance diagram for InterpolatedForwardCurve:



7.300.1 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()`)

7.300.2 Member Function Documentation

7.300.2.1 [Rate](#) zeroYieldImpl ([Time](#) *t*) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

Warning:

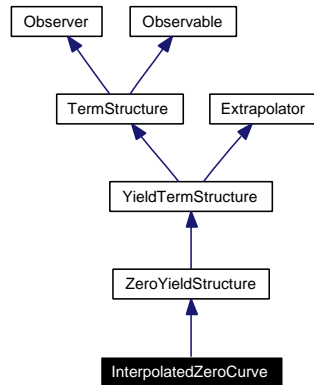
This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Reimplemented from [ForwardRateStructure](#).

7.301 InterpolatedZeroCurve Class Template Reference

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

Inheritance diagram for InterpolatedZeroCurve:



7.301.1 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_`

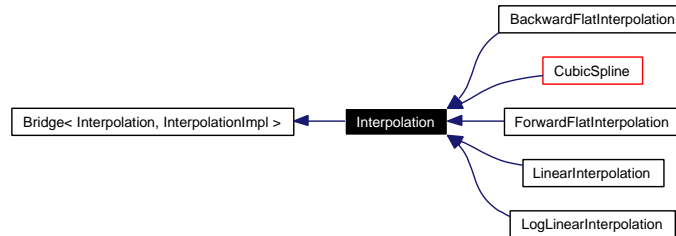
Public Member Functions

- **InterpolatedZeroCurve** (const `std::vector< Date > &dates`, const `std::vector< Rate > &yields`, const `DayCounter &dayCounter`, const `Interpolator &interpolator=Interpolator()`)

7.302 Interpolation Class Reference

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

Inheritance diagram for Interpolation:



7.302.1 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 allowExtrapolation) const

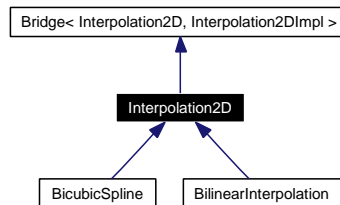
Classes

- class [templateImpl](#)
basic template implementation

7.303 Interpolation2D Class Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2D:



7.303.1 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 `allowExtrapolation`) const

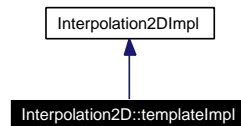
Classes

- class [templateImpl](#)
basic template implementation

7.304 Interpolation2D::templateImpl Class Template Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2D::templateImpl:



7.304.1 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

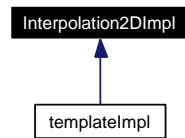
Protected Attributes

- I1 **xBegin_**
- I1 **xEnd_**
- I2 **yBegin_**
- I2 **yEnd_**
- const M & **zData_**

7.305 Interpolation2DImpl Class Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2DImpl:



7.305.1 Detailed Description

abstract base class for 2-D interpolation implementations

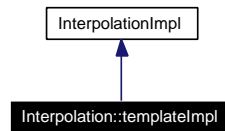
Public Member Functions

- 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

7.306 Interpolation::templateImpl Class Template Reference

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

Inheritance diagram for Interpolation::templateImpl:



7.306.1 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

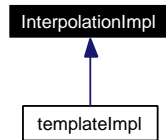
Protected Attributes

- I1 **xBegin_**
- I1 **xEnd_**
- I2 **yBegin_**

7.307 InterpolationImpl Class Reference

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

Inheritance diagram for InterpolationImpl:



7.307.1 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

7.308 InverseCumulativeNormal Class Reference

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

7.308.1 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

7.309 InverseCumulativePoisson Class Reference

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

7.309.1 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

7.310 InverseCumulativeRng Class Template Reference

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

7.310.1 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

7.311 InverseCumulativeRsg Class Template Reference

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

7.311.1 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**

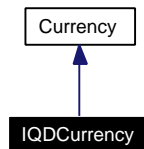
Public Member Functions

- **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

7.312 IQDCurrency Class Reference

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

Inheritance diagram for IQDCurrency:



7.312.1 Detailed Description

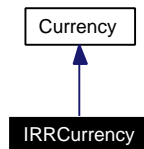
Iraqi dinar.

The ISO three-letter code is IQD; the numeric code is 368. It is divided in 1000 fils.

7.313 IRRCurrency Class Reference

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

Inheritance diagram for IRRCurrency:



7.313.1 Detailed Description

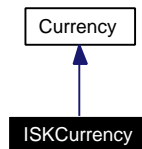
Iranian rial.

The ISO three-letter code is IRR; the numeric code is 364. It has no subdivisions.

7.314 ISKCurrency Class Reference

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

Inheritance diagram for ISKCurrency:



7.314.1 Detailed Description

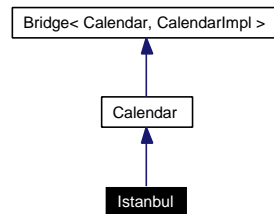
Iceland krona.

The ISO three-letter code is ISK; the numeric code is 352. It is divided in 100 aurar.

7.315 Istanbul Class Reference

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

Inheritance diagram for Istanbul:



7.315.1 Detailed Description

Istanbul calendar

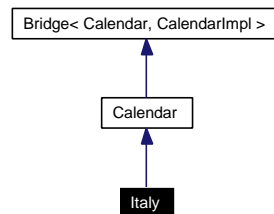
Holidays:

- 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)

7.316 Italy Class Reference

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

Inheritance diagram for Italy:



7.316.1 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)

7.316.2 Member Enumeration Documentation

7.316.2.1 enum [Market](#)

Italian calendars.

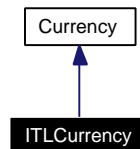
Enumerator:

- Settlement* generic settlement calendar
- Exchange* Milan stock-exchange calendar.

7.317 ITLCurrency Class Reference

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

Inheritance diagram for ITLCurrency:



7.317.1 Detailed Description

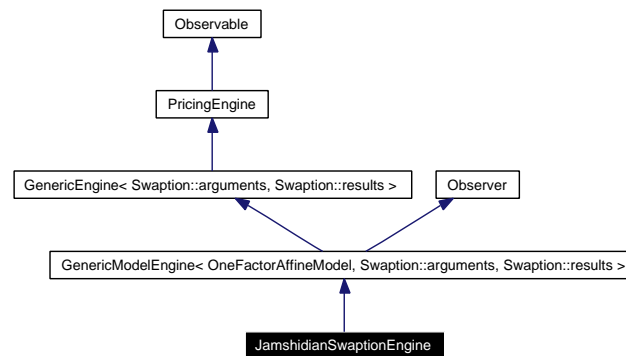
Italian lira.

The ISO three-letter code was ITL; the numeric code was 380. It had no subdivisions.

7.318 JamshidianSwaptionEngine Class Reference

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

Inheritance diagram for JamshidianSwaptionEngine:



7.318.1 Detailed Description

Jamshidian swaption engine.

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- **JamshidianSwaptionEngine** (const boost::shared_ptr< [OneFactorAffineModel](#) > &model)
- void **calculate** () const

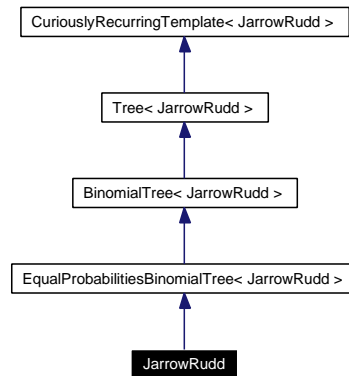
Friends

- class **rStarFinder**

7.319 JarrowRudd Class Reference

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

Inheritance diagram for JarrowRudd:



7.319.1 Detailed Description

Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

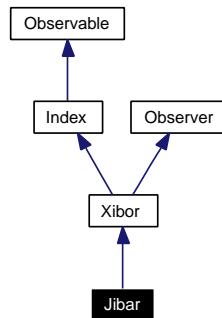
Public Member Functions

- **JarrowRudd** (const boost::shared_ptr< [StochasticProcess1D](#) > &process, [Time](#) end, [Size](#) steps, [Real](#) strike)

7.320 Jibar Class Reference

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

Inheritance diagram for Jibar:



7.320.1 Detailed Description

JIBAR rate

[Johannesburg](#) Interbank Agreed Rate

Todo

check settlement days and day-count convention.

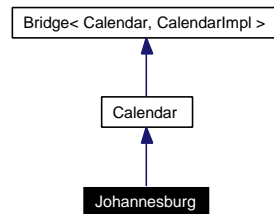
Public Member Functions

- **Jibar**([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [DayCounter](#) &dc=[Actual365Fixed](#)())

7.321 Johannesburg Class Reference

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

Inheritance diagram for Johannesburg:



7.321.1 Detailed Description

Johannesburg calendar

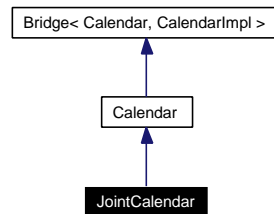
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Family Day, Easter Monday
- Human Rights Day, March 21st (possibly moved to Monday)
- Freedom Day, April 27th (possibly moved to Monday)
- Workers Day, May 1st (possibly moved to Monday)
- Youth Day, June 16th (possibly moved to Monday)
- National Women's Day, August 9th (possibly moved to Monday)
- Heritage Day, September 24th (possibly moved to Monday)
- Day of Reconciliation, December 16th (possibly moved to Monday)
- Christmas December 25th
- Day of Goodwill December 26th (possibly moved to Monday)

7.322 JointCalendar Class Reference

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

Inheritance diagram for JointCalendar:



7.322.1 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.

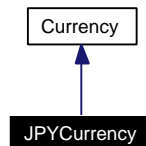
Public Member Functions

- **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)

7.323 JPYCurrency Class Reference

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

Inheritance diagram for JPYCurrency:



7.323.1 Detailed Description

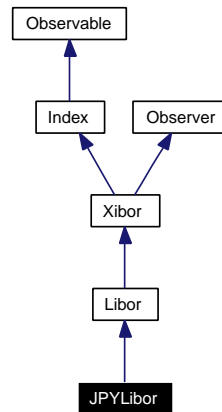
Japanese yen.

The ISO three-letter code is JPY; the numeric code is 392. It is divided into 100 sen.

7.324 JPYLibor Class Reference

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

Inheritance diagram for JPYLibor:



7.324.1 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.

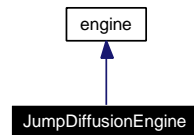
Public Member Functions

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

7.325 JumpDiffusionEngine Class Reference

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

Inheritance diagram for JumpDiffusionEngine:



7.325.1 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.

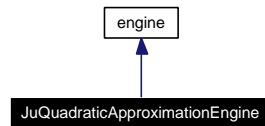
Public Member Functions

- **JumpDiffusionEngine** (const boost::shared_ptr< [VanillaOption::engine](#) > &, [Real](#) relative-Accuracy_=1e-4, Size maxIterations=100)
- void **calculate** () const

7.326 JuQuadraticApproximationEngine Class Reference

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

Inheritance diagram for JuQuadraticApproximationEngine:



7.326.1 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

7.327 KnuthUniformRng Class Reference

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

7.327.1 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

7.327.2 Constructor & Destructor Documentation

7.327.2.1 [KnuthUniformRng](#) (long *seed* = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.327.3 Member Function Documentation

7.327.3.1 [KnuthUniformRng::sample_type](#) `next ()` const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

7.328 KronrodIntegral Class Reference

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

7.328.1 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>

Tests

the correctness of the result is tested by checking it against known good values.

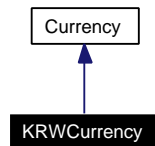
Public Member Functions

- **KronrodIntegral** ([Real](#) tolerance, [Size](#) maxFunctionEvaluations=[Null](#)< [Size](#) >())
- `template<class F> Real operator()` (const F &f, [Real](#) a, [Real](#) b) const
- [Size](#) `functionEvaluations` ()

7.329 KRWCurrency Class Reference

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

Inheritance diagram for KRWCurrency:



7.329.1 Detailed Description

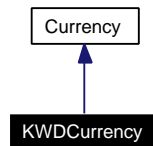
South-Korean won.

The ISO three-letter code is KRW; the numeric code is 410. It is divided in 100 chon.

7.330 KWDCurrency Class Reference

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

Inheritance diagram for KWDCurrency:



7.330.1 Detailed Description

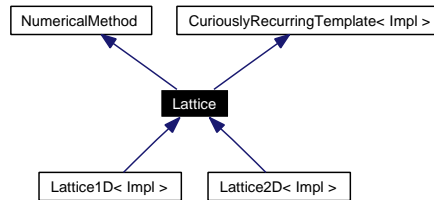
Kuwaiti dinar.

The ISO three-letter code is KWD; the numeric code is 414. It is divided in 1000 fils.

7.331 Lattice Class Template Reference

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

Inheritance diagram for Lattice:



7.331.1 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_`

7.331.2 Member Function Documentation

7.331.2.1 void `rollback` ([DiscretizedAsset](#) &, [Time](#) *to*) const [virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implements [NumericalMethod](#).

7.331.2.2 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:

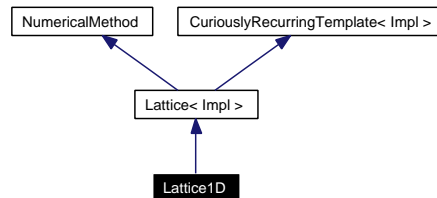
```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Implements [NumericalMethod](#).

7.332 Lattice1D Class Template Reference

```
#include <ql/Lattices/lattice1d.hpp>
```

Inheritance diagram for Lattice1D:



7.332.1 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;
```

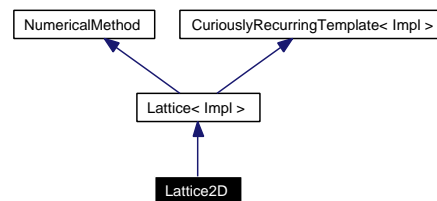
Public Member Functions

- **Lattice1D** (const [TimeGrid](#) &timeGrid, [Size](#) n)
- **Disposable**< [Array](#) > **grid** ([Time](#) t) const
- **Real** **underlying** ([Size](#) i, [Size](#) index) const

7.333 Lattice2D Class Template Reference

```
#include <ql/Lattices/lattice2d.hpp>
```

Inheritance diagram for Lattice2D:



7.333.1 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

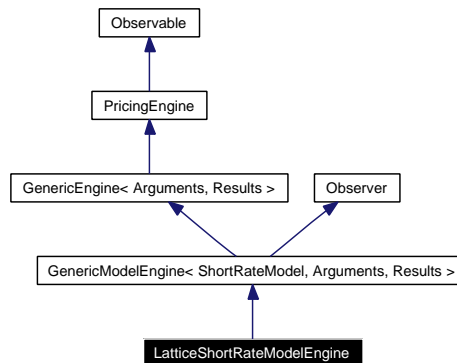
Protected Attributes

- boost::shared_ptr< T > **tree1_**
- boost::shared_ptr< T > **tree2_**

7.334 LatticeShortRateModelEngine Class Template Reference

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

Inheritance diagram for LatticeShortRateModelEngine:



7.334.1 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_

7.334.2 Member Function Documentation

7.334.2.1 void update () [virtual]

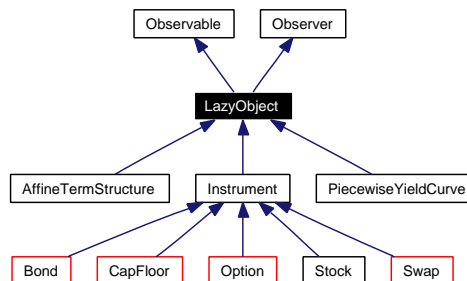
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [GenericModelEngine< ShortRateModel, Arguments, Results >](#).

7.335 LazyObject Class Reference

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

Inheritance diagram for LazyObject:



7.335.1 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_`

7.335.2 Member Function Documentation

7.335.2.1 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](#).

7.335.2.2 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.

7.335.2.3 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.

7.335.2.4 void unfreeze ()

This method reverts the effect of the **freeze** method, thus re-enabling recalculations.

7.335.2.5 void calculate () const [protected, virtual]

This method performs all needed calculations by calling the **performCalculations** method.

Warning:

Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.

Should this method be redefined in derived classes, [LazyObject::calculate\(\)](#) should be called in the overriding method.

Reimplemented in [Instrument](#).

7.335.2.6 virtual void performCalculations () const [protected, pure virtual]

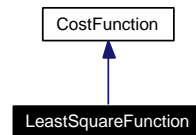
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implemented in [Instrument](#), [BarrierOption](#), [Bond](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), [Stock](#), and [Swap](#).

7.336 LeastSquareFunction Class Reference

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

Inheritance diagram for LeastSquareFunction:



7.336.1 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

7.337 LeastSquareProblem Class Reference

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

7.337.1 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

7.337.2 Member Function Documentation

7.337.2.1 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

7.338 LecuyerUniformRng Class Reference

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

7.338.1 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

7.338.2 Constructor & Destructor Documentation

7.338.2.1 [LecuyerUniformRng](#) (long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.338.3 Member Function Documentation

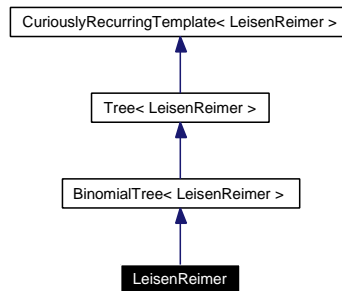
7.338.3.1 [sample_type next](#) () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

7.339 LeisenReimer Class Reference

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

Inheritance diagram for LeisenReimer:



7.339.1 Detailed Description

Leisen & Reimer tree: multiplicative approach.

Public Member Functions

- **LeisenReimer** (const boost::shared_ptr< [StochasticProcess1D](#) > &process, [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_

7.340 LexicographicalView Class Template Reference

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

7.340.1 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)

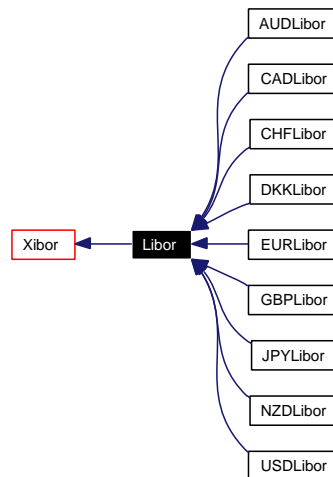
Inspectors

- [Size xSize \(\)](#) const
dimension of the array along x
- [Size ySize \(\)](#) const
dimension of the array along y

7.341 Libor Class Reference

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

Inheritance diagram for Libor:



7.341.1 Detailed Description

base class for BBA LIBOR indexes

Public Member Functions

- **Libor** (const std::string &familyName, Integer n, TimeUnit units, Integer settlementDays, const Currency ¤cy, const Calendar &localCalendar, const Calendar ¤cyCalendar, 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

7.342 Linear Class Reference

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

7.342.1 Detailed Description

[Linear](#) interpolation factory and traits.

Public Types

- enum { **global** = 0 }

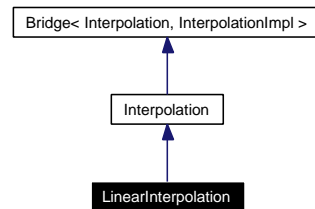
Public Member Functions

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

7.343 LinearInterpolation Class Reference

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

Inheritance diagram for LinearInterpolation:



7.343.1 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)`

7.343.2 Constructor & Destructor Documentation

7.343.2.1 [LinearInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

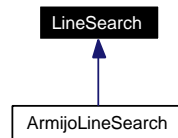
Precondition:

the *x* values must be sorted.

7.344 LineSearch Class Reference

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

Inheritance diagram for LineSearch:



7.344.1 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](#) ¶ms, const [Array](#) &direction, [Real](#) beta, const [Constraint](#) &constraint)

Protected Attributes

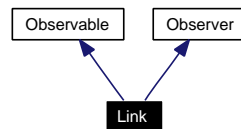
- [Array](#) [xtd_](#)
new x and its gradient
- [Array](#) [gradient_](#)

- [Real qt_](#)
cost function value and gradient norm corresponding to xtd_
- [Real qpt_](#)
- [bool succeed_](#)
flag to know if linesearch succeed

7.345 Link Class Template Reference

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

Inheritance diagram for Link:



7.345.1 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.

7.345.2 Constructor & Destructor Documentation

7.345.2.1 [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.

7.345.3 Member Function Documentation

7.345.3.1 `void linkTo (const boost::shared_ptr< Type > &, bool registerAsObserver = true)`

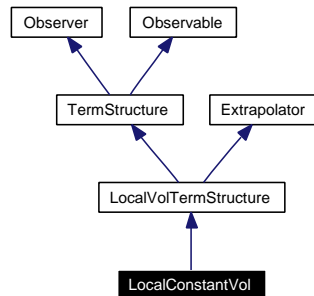
Warning:

`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.

7.346 LocalConstantVol Class Reference

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

Inheritance diagram for LocalConstantVol:



7.346.1 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

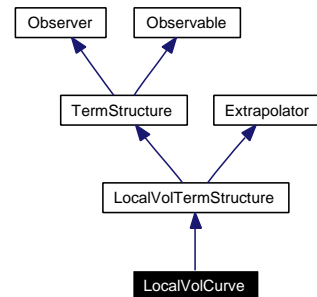
Visitability

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

7.347 LocalVolCurve Class Reference

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

Inheritance diagram for LocalVolCurve:



7.347.1 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

7.347.2 Member Function Documentation

7.347.2.1 [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

$$\sigma_L(t) = \sqrt{\frac{d}{dt} \sigma_B^2(t) t}$$

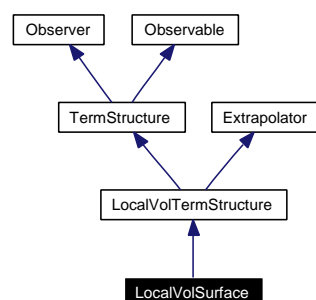
can be deduced which is here implemented.

Implements [LocalVolTermStructure](#).

7.348 LocalVolSurface Class Reference

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

Inheritance diagram for LocalVolSurface:



7.348.1 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

Bug

this class is untested, probably unreliable.

Public Member Functions

- **LocalVolSurface** (const [Handle](#)< [BlackVolTermStructure](#) > &blackTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, const [Handle](#)< [Quote](#) > &underlying)
- **LocalVolSurface** (const [Handle](#)< [BlackVolTermStructure](#) > &blackTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, [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](#) &)

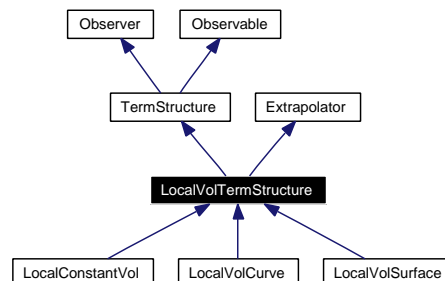
Protected Member Functions

- [Volatility](#) `localVolImpl` ([Time](#), [Real](#)) const
local vol calculation

7.349 LocalVolTermStructure Class Reference

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

Inheritance diagram for LocalVolTermStructure:



7.349.1 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

7.349.2 Constructor & Destructor Documentation

7.349.2.1 [LocalVolTermStructure](#) ()

default constructor

Warning:

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

7.350 LogLinear Class Reference

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

7.350.1 Detailed Description

log-linear interpolation factory and traits

Public Types

- enum { **global** = 0 }

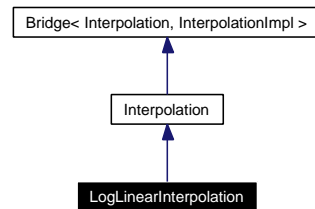
Public Member Functions

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

7.351 LogLinearInterpolation Class Reference

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

Inheritance diagram for LogLinearInterpolation:



7.351.1 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)`

7.351.2 Constructor & Destructor Documentation

7.351.2.1 [LogLinearInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

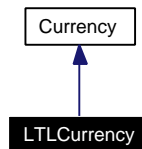
Precondition:

the *x* values must be sorted.

7.352 LTLCurrency Class Reference

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

Inheritance diagram for LTLCurrency:



7.352.1 Detailed Description

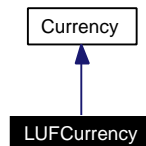
Lithuanian litas.

The ISO three-letter code is LTL; the numeric code is 440. It is divided in 100 centu.

7.353 LUFCurrency Class Reference

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

Inheritance diagram for LUFCurrency:



7.353.1 Detailed Description

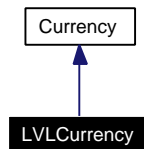
Luxembourg franc.

The ISO three-letter code is LUF; the numeric code is 442. It is divided in 100 centimes.

7.354 LVLCurrency Class Reference

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

Inheritance diagram for LVLCurrency:



7.354.1 Detailed Description

Latvian lat.

The ISO three-letter code is LVL; the numeric code is 428. It is divided in 100 santims.

7.355 MakeMCDigitalEngine Class Template Reference

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

7.355.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MakeMCDigitalEngine< RNG, S >

Monte Carlo digital engine factory.

Public Member Functions

- [MakeMCDigitalEngine](#) & **withSteps** ([Size](#) steps)
- [MakeMCDigitalEngine](#) & **withStepsPerYear** ([Size](#) steps)
- [MakeMCDigitalEngine](#) & **withBrownianBridge** (bool b=true)
- [MakeMCDigitalEngine](#) & **withSamples** ([Size](#) samples)
- [MakeMCDigitalEngine](#) & **withTolerance** ([Real](#) tolerance)
- [MakeMCDigitalEngine](#) & **withMaxSamples** ([Size](#) samples)
- [MakeMCDigitalEngine](#) & **withSeed** (BigNatural seed)
- [MakeMCDigitalEngine](#) & **withAntitheticVariate** (bool b=true)
- [MakeMCDigitalEngine](#) & **withControlVariate** (bool b=true)
- **operator boost::shared_ptr () const**

7.356 MakeMCEuropeanEngine Class Template Reference

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

7.356.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::Make-  
MCEuropeanEngine< RNG, S >
```

Monte Carlo European engine factory.

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`

7.357 MakeMCEuropeanHestonEngine Class Template Reference

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

7.357.1 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

7.358 MakeSchedule Class Reference

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

7.358.1 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** ()

7.359 Matrix Class Reference

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

7.359.1 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.

Examples:

[AmericanOption.cpp](#), and [EuropeanOption.cpp](#).

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`)

7.359.2 Member Function Documentation

7.359.2.1 const `Matrix` & operator+= (const `Matrix` &)

Precondition:

all matrices involved in an algebraic expression must have the same size.

7.359.3 Friends And Related Function Documentation

7.359.3.1 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.

7.359.3.2 `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real componentRetainedPercentage, SalvagingAlgorithm::Type)` [related]

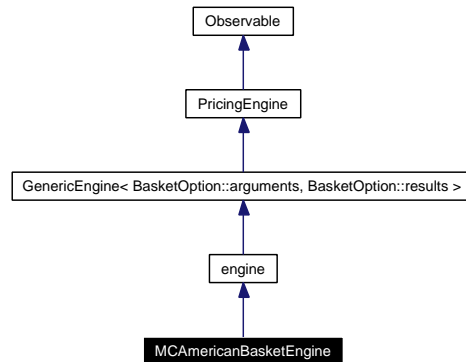
Precondition:

the given matrix must be symmetric.

7.360 MCAmericanBasketEngine Class Reference

```
#include <ql/PricingEngines/Basket/mcamericanbasketengine.hpp>
```

Inheritance diagram for MCAmericanBasketEngine:



7.360.1 Detailed Description

least-square Monte Carlo engine

Warning:

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.

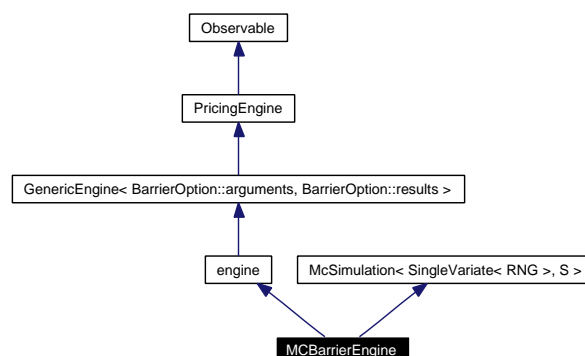
Public Member Functions

- **MCAmericanBasketEngine** ([Size](#) requiredSamples, [Size](#) timeSteps, BigNatural seed=0)
- void **calculate** () const

7.361 MBarrierEngine Class Template Reference

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

Inheritance diagram for MBarrierEngine:



7.361.1 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`

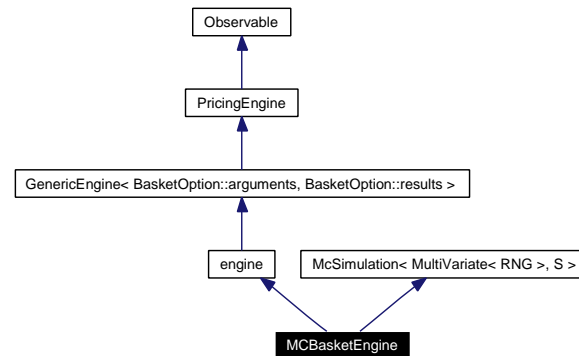
Protected Attributes

- `Size maxTimeStepsPerYear_`
- `Size requiredSamples_`
- `Size maxSamples_`
- `Real requiredTolerance_`
- `bool isBiased_`
- `bool brownianBridge_`
- `BigNatural seed_`

7.362 MCBasketEngine Class Template Reference

```
#include <ql/PricingEngines/Basket/mcbasketengine.hpp>
```

Inheritance diagram for MCBasketEngine:



7.362.1 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

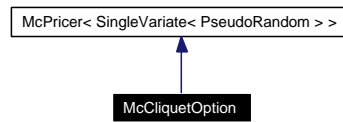
Protected Attributes

- [Size](#) `maxTimeStepsPerYear_`
- [Size](#) `requiredSamples_`
- [Size](#) `maxSamples_`
- [Real](#) `requiredTolerance_`
- `bool brownianBridge_`
- `BigNatural seed_`

7.363 McCliquetOption Class Reference

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

Inheritance diagram for McCliquetOption:



7.363.1 Detailed Description

simple example of Monte Carlo pricer

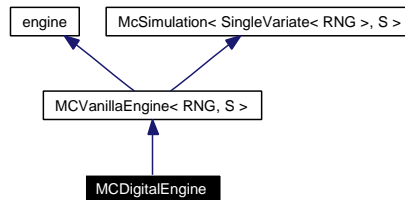
Public Member Functions

- **McCliquetOption** (Option::Type type, [Real](#) underlying, [Real](#) moneyness, const [Handle](#)<[YieldTermStructure](#)> ÷ndYield, const [Handle](#)<[YieldTermStructure](#)> &riskFreeRate, const [Handle](#)<[BlackVolTermStructure](#)> &volatility, const std::vector<[Time](#)> ×, [Real](#) accruedCoupon, [Real](#) lastFixing, [Real](#) localCap, [Real](#) localFloor, [Real](#) globalCap, [Real](#) globalFloor, bool redemptionOnly, [BigNatural](#) seed=0)

7.364 MCDigitalEngine Class Template Reference

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

Inheritance diagram for MCDigitalEngine:



7.364.1 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](#)< RNG, S >::path_generator_type **path_generator_type**
- typedef [MCVanillaEngine](#)< RNG, S >::path_pricer_type **path_pricer_type**
- typedef [MCVanillaEngine](#)< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCDigitalEngine** ([Size](#) timeSteps, [Size](#) timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, BigNatural seed)

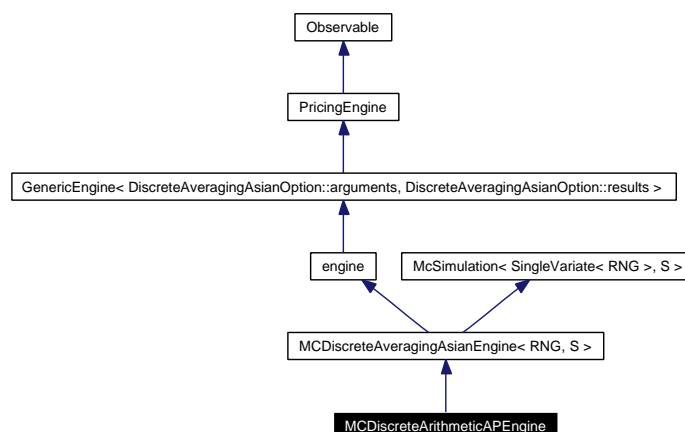
Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.365 MCDiscreteArithmeticAPEngine Class Template Reference

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

Inheritance diagram for MCDiscreteArithmeticAPEngine:



7.365.1 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](#))

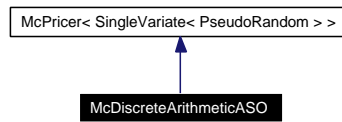
Protected Member Functions

- `boost::shared_ptr< path_pricer_type > pathPricer () const`
- `boost::shared_ptr< path_pricer_type > controlPathPricer () const`
- `boost::shared_ptr< PricingEngine > controlPricingEngine () const`

7.366 McDiscreteArithmeticASO Class Reference

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

Inheritance diagram for McDiscreteArithmeticASO:



7.366.1 Detailed Description

example of Monte Carlo pricer using a control variate.

Todo

continous-averaging version

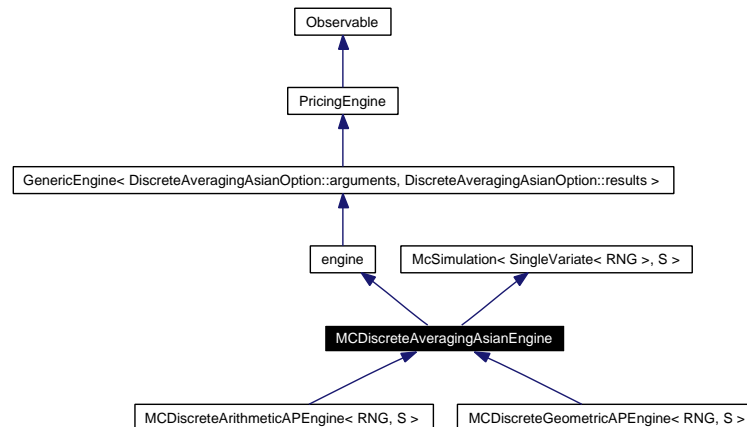
Public Member Functions

- **McDiscreteArithmeticASO** (Option::Type type, [Real](#) underlying, const [Handle](#)< [YieldTermStructure](#) > ÷ndYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > ×, bool controlVariate, BigNatural seed=0)

7.367 MCDiscreteAveragingAsianEngine Class Template Reference

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

Inheritance diagram for MCDiscreteAveragingAsianEngine:



7.367.1 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`

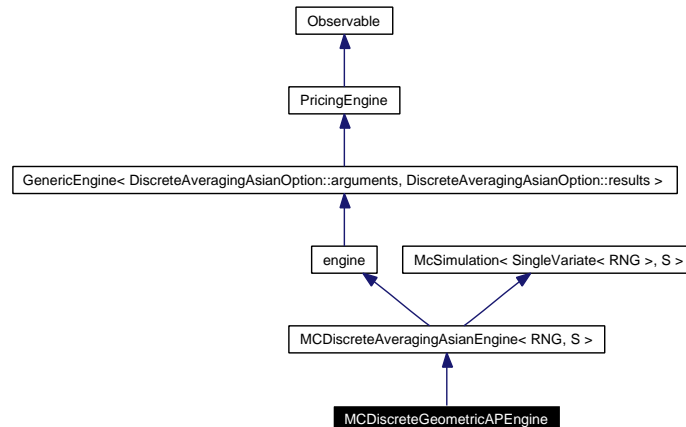
Protected Attributes

- `Size maxTimeStepsPerYear_`
- `Size requiredSamples_`
- `Size maxSamples_`
- `Real requiredTolerance_`
- `bool brownianBridge_`
- `BigNatural seed_`

7.368 MCDiscreteGeometricAPEngine Class Template Reference

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

Inheritance diagram for MCDiscreteGeometricAPEngine:



7.368.1 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)

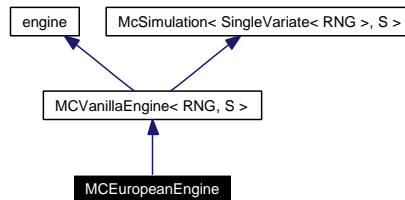
Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.369 MCEuropeanEngine Class Template Reference

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

Inheritance diagram for MCEuropeanEngine:



7.369.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCEuropean-
Engine< RNG, S >
```

European option pricing engine using Monte Carlo simulation.

Tests

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

Public Types

- typedef [MCVanillaEngine< RNG, S >::path_generator_type](#) **path_generator_type**
- typedef [MCVanillaEngine< RNG, S >::path_pricer_type](#) **path_pricer_type**
- typedef [MCVanillaEngine< RNG, S >::stats_type](#) **stats_type**

Public Member Functions

- **MCEuropeanEngine** ([Size](#) timeSteps, [Size](#) timeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, BigNatural seed)

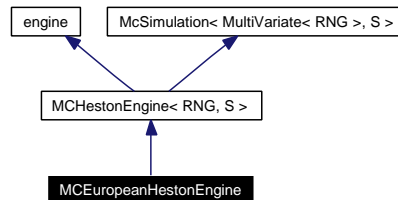
Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.370 MCEuropeanHestonEngine Class Template Reference

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

Inheritance diagram for MCEuropeanHestonEngine:



7.370.1 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 [MCHestonEngine](#)< 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)

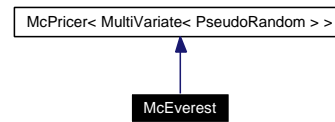
Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.371 McEverest Class Reference

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

Inheritance diagram for McEverest:



7.371.1 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

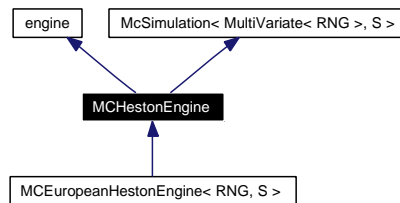
Public Member Functions

- **McEverest** (const std::vector< [Handle< YieldTermStructure >](#) ÷ndYield, const [Handle< YieldTermStructure >](#) &riskFreeRate, const std::vector< [Handle< BlackVolTermStructure >](#) &volatilities, const [Matrix](#) &correlation, [Time](#) residualTime, [BigNatural](#) seed=0)

7.372 MCHestonEngine Class Template Reference

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

Inheritance diagram for MCHestonEngine:



7.372.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCHeston-
Engine< RNG, S >
```

Monte Carlo Heston-model engine.

Public Member Functions

- void **calculate** () const

Protected 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**

Protected Member Functions

- **MCHestonEngine** ([Size](#) timeSteps, [Size](#) timeStepsPerYear, bool antitheticVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, BigNatural seed)
- [TimeGrid](#) **timeGrid** () const
- boost::shared_ptr< path_generator_type > **pathGenerator** () const

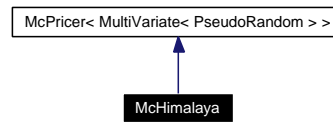
Protected Attributes

- [Size](#) timeSteps_
- [Size](#) timeStepsPerYear_
- [Size](#) requiredSamples_
- [Size](#) maxSamples_
- [Real](#) requiredTolerance_
- bool brownianBridge_
- BigNatural seed_

7.373 McHimalaya Class Reference

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

Inheritance diagram for McHimalaya:



7.373.1 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.

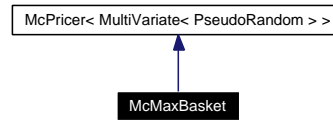
Public Member Functions

- **McHimalaya** (const std::vector< [Real](#) > &underlyings, const std::vector< [Handle](#)< [YieldTermStructure](#) > > ÷ndYields, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const std::vector< [Handle](#)< [BlackVolTermStructure](#) > > &volatilities, const [Matrix](#) &correlation, [Real](#) strike, const std::vector< [Time](#) > ×, BigNatural seed=0)

7.374 McMaxBasket Class Reference

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

Inheritance diagram for McMaxBasket:



7.374.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

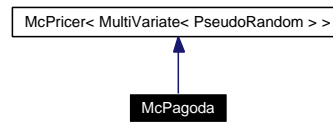
Public Member Functions

- **McMaxBasket** (const std::vector< [Real](#) > &underlyings, const std::vector< [Handle](#)< [YieldTermStructure](#) > > ÷ndYields, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const std::vector< [Handle](#)< [BlackVolTermStructure](#) > > &volatilities, const [Matrix](#) &correlation, [Time](#) residualTime, BigNatural seed=0)

7.375 McPagoda Class Reference

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

Inheritance diagram for McPagoda:



7.375.1 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.

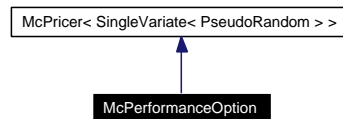
Public Member Functions

- **McPagoda** (const std::vector< Real > &underlyings, Real fraction, Real roof, const std::vector< Handle< YieldTermStructure > > ÷ndYields, const Handle< YieldTermStructure > &riskFreeRate, const std::vector< Handle< BlackVolTermStructure > > &volatilities, const Matrix &correlation, const std::vector< Time > ×, BigNatural seed=0)

7.376 McPerformanceOption Class Reference

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

Inheritance diagram for McPerformanceOption:



7.376.1 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](#) > ÷ndYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > ×, [BigNatural](#) seed=0)

7.377 McPricer Class Template Reference

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

7.377.1 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

7.378 McSimulation Class Template Reference

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

7.378.1 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 McEngine gives an easy way to write a Monte Carlo engine.

See McVanillaEngine as an example of one factor engine.

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

7.378.2 Member Function Documentation

7.378.2.1 void calculate ([Real](#) *requiredTolerance*, [Size](#) *requiredSamples*, [Size](#) *maxSamples*)
const

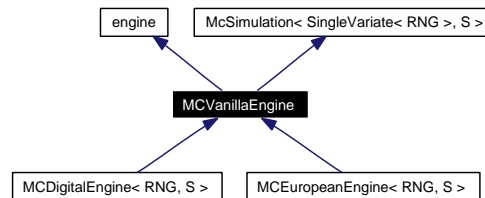
basic calculate method provided to inherited pricing engines

Initialize the one-factor Monte Carlo

7.379 MCVanillaEngine Class Template Reference

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

Inheritance diagram for MCVanillaEngine:



7.379.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCVanillaEngine< RNG, S >
```

Pricing engine for vanilla options using Monte Carlo simulation.

Public Member Functions

- void **calculate** () const

Protected 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**

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_**

7.380 MersenneTwisterUniformRng Class Reference

```
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

7.380.1 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

7.380.2 Constructor & Destructor Documentation

7.380.2.1 [MersenneTwisterUniformRng](#) (unsigned long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.380.3 Member Function Documentation

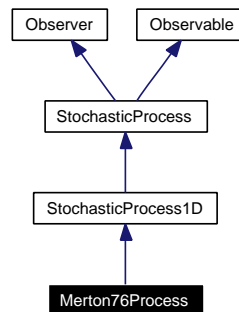
7.380.3.1 [sample_type](#) next () const

returns a sample with weight 1.0 containing a random number on (0.0, 1.0)-real-interval

7.381 Merton76Process Class Reference

```
#include <ql/Processes/merton76process.hpp>
```

Inheritance diagram for Merton76Process:



7.381.1 Detailed Description

Merton-76 jump-diffusion process.

Public Member Functions

- **Merton76Process** (const [Handle](#)< [Quote](#) > &stateVariable, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, 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

7.381.2 Member Function Documentation

7.381.2.1 **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](#).

7.381.2.2 **Time** time (const **Date** &) const [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

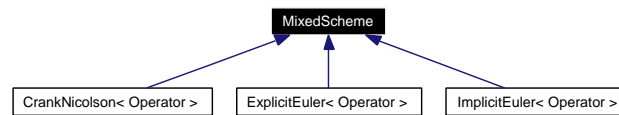
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

7.382 MixedScheme Class Template Reference

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

Inheritance diagram for MixedScheme:



7.382.1 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_

7.383 Money Class Reference

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

7.383.1 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` ¤cy, `Decimal` value)
- `Money` (`Decimal` value, const `Currency` ¤cy)

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)
- std::ostream & operator<< (std::ostream &, const **Money** &)

7.383.2 Member Enumeration Documentation

7.383.2.1 enum **ConversionType**

Enumerator:

NoConversion do not perform conversions

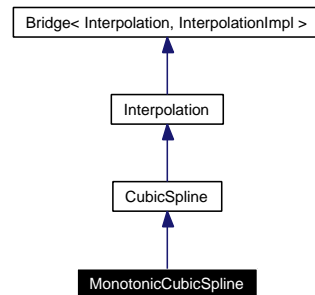
BaseCurrencyConversion convert both operands to the base currency before converting

AutomatedConversion return the result in the currency of the first operand

7.384 MonotonicCubicSpline Class Reference

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

Inheritance diagram for MonotonicCubicSpline:



7.384.1 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)`

7.384.2 Constructor & Destructor Documentation

- 7.384.2.1 `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.

7.385 MonteCarloModel Class Template Reference

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

7.385.1 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

7.386 MoreGreeks Class Reference

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

Inheritance diagram for MoreGreeks:



7.386.1 Detailed Description

more additional option results

Public Member Functions

- void reset ()

Public Attributes

- [Real](#) itmCashProbability
- [Real](#) deltaForward
- [Real](#) elasticity
- [Real](#) thetaPerDay
- [Real](#) strikeSensitivity

7.387 MoroInverseCumulativeNormal Class Reference

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

7.387.1 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](#)

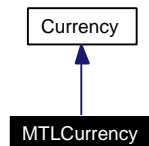
Public Member Functions

- **MoroInverseCumulativeNormal** ([Real](#) average=0.0, [Real](#) sigma=1.0)
- **Real operator()** ([Real](#) x) const

7.388 MTLCurrency Class Reference

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

Inheritance diagram for MTLCurrency:



7.388.1 Detailed Description

Maltese lira.

The ISO three-letter code is MTL; the numeric code is 470. It is divided in 100 cents.

7.389 MultiAsset Struct Template Reference

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

7.389.1 Detailed Description

```
template<class rng_traits = PseudoRandom> struct QuantLib::MultiAsset< rng_traits >
```

Deprecated

use [MultiVariate](#) instead

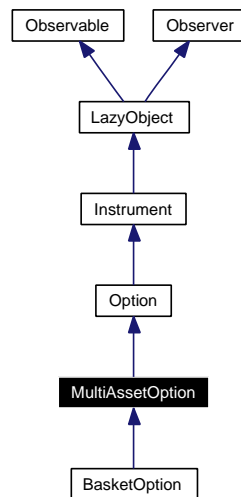
Public Types

- typedef [MultiVariate](#)< rng_traits >::path_type path_type
- typedef [MultiVariate](#)< rng_traits >::path_pricer_type path_pricer_type
- typedef [MultiVariate](#)< rng_traits >::rsg_type rsg_type
- typedef [MultiVariate](#)< rng_traits >::path_generator_type path_generator_type

7.390 MultiAssetOption Class Reference

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

Inheritance diagram for MultiAssetOption:



7.390.1 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

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
- void [performCalculations](#) () 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

7.390.2 Member Function Documentation

7.390.2.1 `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](#).

7.390.2.2 `void setupExpired () const` [protected, virtual]

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

Reimplemented from [Instrument](#).

7.390.2.3 `void performCalculations () const` [protected, virtual]

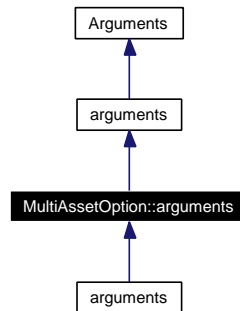
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

7.391 MultiAssetOption::arguments Class Reference

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

Inheritance diagram for MultiAssetOption::arguments:



7.391.1 Detailed Description

Arguments for multi-asset option calculation

Public Member Functions

- `void validate () const`

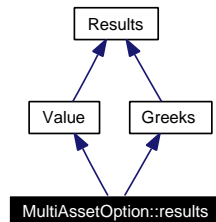
Public Attributes

- `boost::shared_ptr< StochasticProcess > stochasticProcess`

7.392 MultiAssetOption::results Class Reference

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

Inheritance diagram for MultiAssetOption::results:



7.392.1 Detailed Description

Results from multi-asset option calculation

Public Member Functions

- `void reset ()`

7.393 MultiCubicSpline Class Template Reference

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

7.393.1 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

7.394 MultiPath Class Reference

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

7.394.1 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)`

7.395 MultiPathGenerator Class Template Reference

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

7.395.1 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

7.396 MultiVariate Struct Template Reference

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

7.396.1 Detailed Description

template<class rng_traits = PseudoRandom> struct QuantLib::MultiVariate< rng_traits >

default Monte Carlo traits for multi-variate models

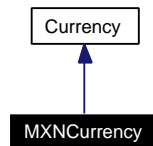
Public Types

- typedef [MultiPath](#) **path_type**
- typedef [PathPricer](#)< [path_type](#) > **path_pricer_type**
- typedef rng_traits::rsg_type **rsg_type**
- typedef [MultiPathGenerator](#)< rsg_type > **path_generator_type**

7.397 MXNCurrency Class Reference

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

Inheritance diagram for MXNCurrency:



7.397.1 Detailed Description

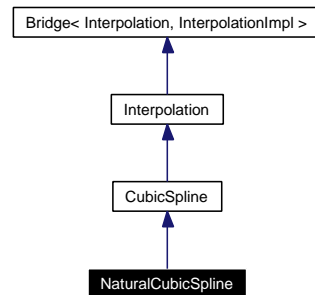
Mexican peso.

The ISO three-letter code is MXN; the numeric code is 484. It is divided in 100 centavos.

7.398 NaturalCubicSpline Class Reference

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

Inheritance diagram for NaturalCubicSpline:



7.398.1 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)`

7.398.2 Constructor & Destructor Documentation

7.398.2.1 [NaturalCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

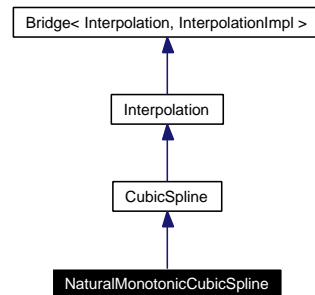
Precondition:

the *x* values must be sorted.

7.399 NaturalMonotonicCubicSpline Class Reference

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

Inheritance diagram for NaturalMonotonicCubicSpline:



7.399.1 Detailed Description

Natural cubic spline with monotonicity constraint.

Public Member Functions

- `template<class I1, class I2> NaturalMonotonicCubicSpline (const I1 &xBegin, const I1 &x-End, const I2 &yBegin)`

7.399.2 Constructor & Destructor Documentation

7.399.2.1 [NaturalMonotonicCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

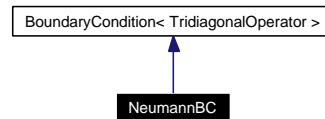
Precondition:

the *x* values must be sorted.

7.400 NeumannBC Class Reference

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

Inheritance diagram for NeumannBC:



7.400.1 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](#))

7.400.2 Member Function Documentation

7.400.2.1 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 >](#).

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

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

7.400.2.3 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 >](#).

7.400.2.4 void applyAfterSolving (Array &) const [virtual]

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

Implements [BoundaryCondition< TridiagonalOperator >](#).

7.400.2.5 void setTime (Time) [virtual]

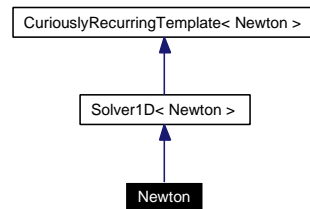
This method sets the current time for time-dependent boundary conditions.

Implements [BoundaryCondition< TridiagonalOperator >](#).

7.401 Newton Class Reference

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

Inheritance diagram for Newton:



7.401.1 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.

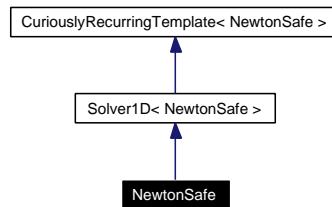
Public Member Functions

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

7.402 NewtonSafe Class Reference

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

Inheritance diagram for NewtonSafe:



7.402.1 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.

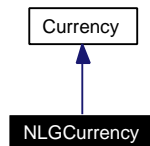
Public Member Functions

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

7.403 NLGCurrency Class Reference

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

Inheritance diagram for NLGCurrency:



7.403.1 Detailed Description

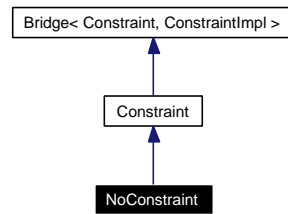
Dutch guilder.

The ISO three-letter code is NLG; the numeric code is 528. It is divided in 100 cents.

7.404 NoConstraint Class Reference

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

Inheritance diagram for NoConstraint:



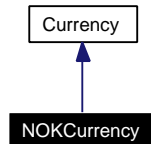
7.404.1 Detailed Description

No constraint.

7.405 NOKCurrency Class Reference

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

Inheritance diagram for NOKCurrency:



7.405.1 Detailed Description

Norwegian krone.

The ISO three-letter code is NOK; the numeric code is 578. It is divided in 100 øre.

7.406 NonLinearLeastSquare Class Reference

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

7.406.1 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 ϕ 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

7.407 NormalDistribution Class Reference

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

7.407.1 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.

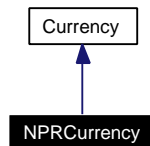
Public Member Functions

- **NormalDistribution** ([Real](#) average=0.0, [Real](#) sigma=1.0)
- **Real operator()** ([Real](#) x) const
- **Real derivative** ([Real](#) x) const

7.408 NPRCurrency Class Reference

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

Inheritance diagram for NPRCurrency:



7.408.1 Detailed Description

Nepal rupee.

The ISO three-letter code is NPR; the numeric code is 524. It is divided in 100 paise.

7.409 Null Class Template Reference

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

7.409.1 Detailed Description

```
template<class Type> class QuantLib::Null< Type >
```

template class providing a null value for a given type.

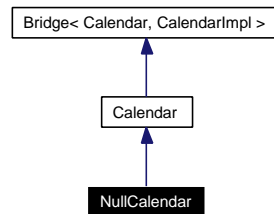
Public Member Functions

- `operator Type () const`

7.410 NullCalendar Class Reference

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

Inheritance diagram for NullCalendar:



7.410.1 Detailed Description

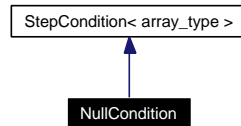
Calendar for reproducing theoretical calculations.

This calendar has no holidays. It ensures that dates at whole-month distances have the same day of month.

7.411 NullCondition Class Template Reference

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

Inheritance diagram for NullCondition:



7.411.1 Detailed Description

```
template<class array_type> class QuantLib::NullCondition< array_type >
```

null step condition

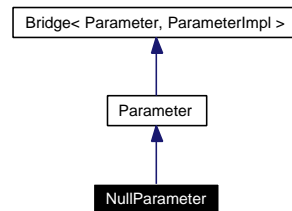
Public Member Functions

- void **applyTo** (array_type &, [Time](#)) const

7.412 NullParameter Class Reference

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

Inheritance diagram for NullParameter:



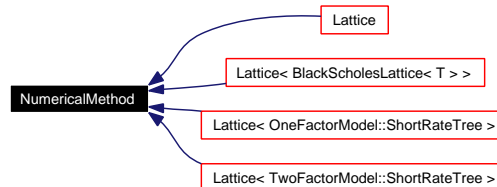
7.412.1 Detailed Description

Parameter which is always zero $a(t) = 0$

7.413 NumericalMethod Class Reference

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

Inheritance diagram for NumericalMethod:



7.413.1 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_

7.413.2 Member Function Documentation

7.413.2.1 virtual void rollback ([DiscretizedAsset](#) &, [Time](#) to) const [pure virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implemented in [Lattice](#), [Lattice< OneFactorModel::ShortRateTree >](#), [Lattice< TwoFactorModel::ShortRateTree >](#), and [Lattice< BlackScholesLattice< T > >](#).

7.413.2.2 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:

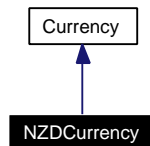
```
method->partialRollback(asset,t);  
asset->preAdjustValues();
```

Implemented in [Lattice](#), [Lattice< OneFactorModel::ShortRateTree >](#), [Lattice< TwoFactorModel::ShortRateTree >](#), and [Lattice< BlackScholesLattice< T > >](#).

7.414 NZDCurrency Class Reference

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

Inheritance diagram for NZDCurrency:



7.414.1 Detailed Description

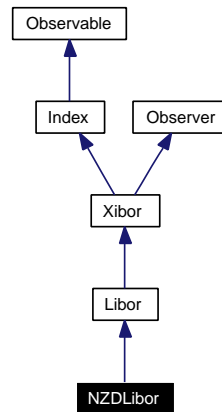
New Zealand dollar.

The ISO three-letter code is NZD; the numeric code is 554. It is divided in 100 cents.

7.415 NZDLibor Class Reference

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

Inheritance diagram for NZDLibor:



7.415.1 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>>.

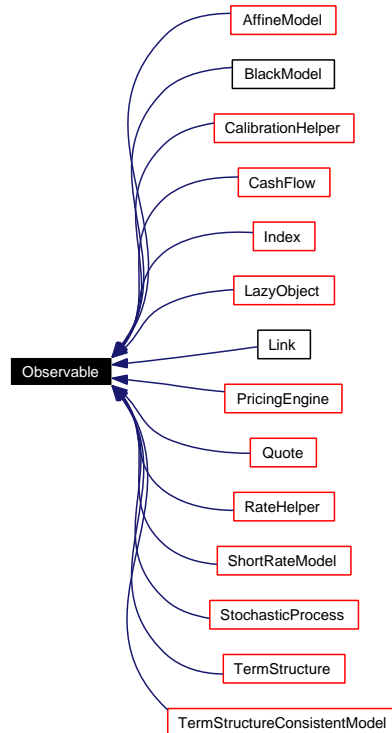
Public Member Functions

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

7.416 Observable Class Reference

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

Inheritance diagram for Observable:



7.416.1 Detailed Description

Object that notifies its changes to a set of observables.

Public Member Functions

- void [notifyObservers](#) ()

Friends

- class `Observer`

7.416.2 Member Function Documentation

7.416.2.1 void notifyObservers ()

This method should be called at the end of non-const methods or when the programmer desires to notify any changes.

7.417 ObservableValue Class Template Reference

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

7.417.1 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

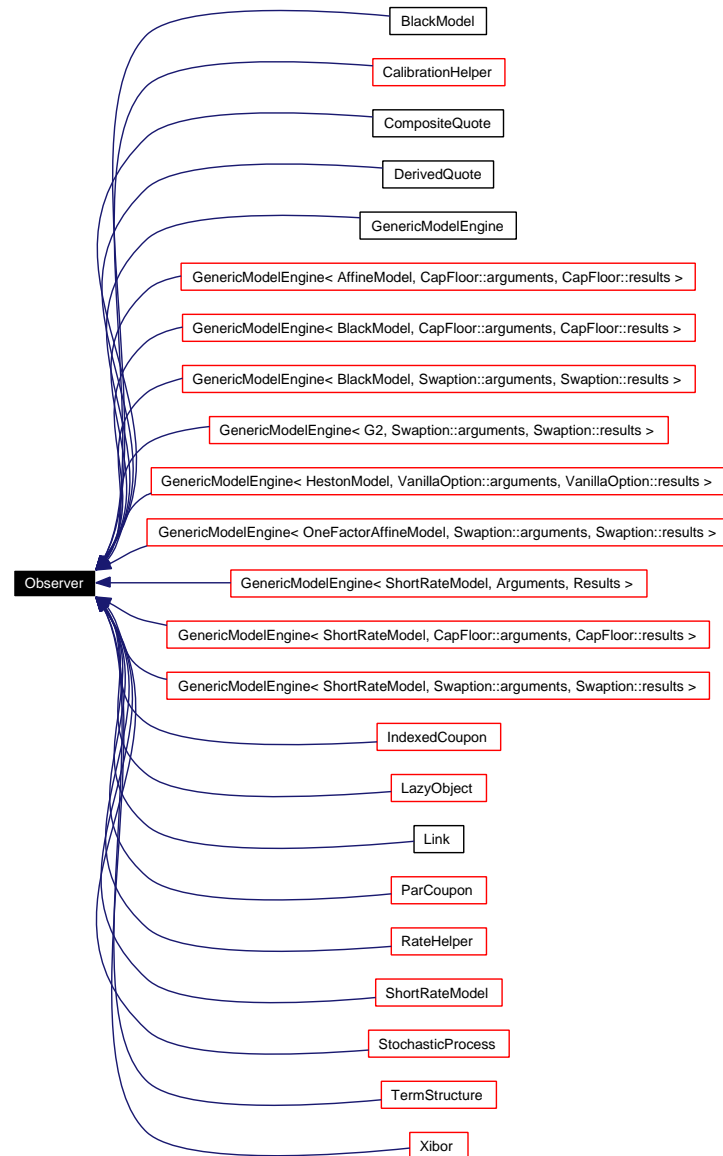
controlled assignment

- **ObservableValue**< T > & **operator=** (const T &)
- **ObservableValue**< T > & **operator=** (const **ObservableValue**< T > &)

7.418 Observer Class Reference

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

Inheritance diagram for Observer:



7.418.1 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

7.418.2 Member Function Documentation

7.418.2.1 virtual void update () [pure virtual]

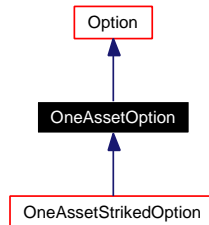
This method must be implemented in derived classes. An instance of [Observer](#) does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

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< 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 >](#), and [LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >](#).

7.419 OneAssetOption Class Reference

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

Inheritance diagram for OneAssetOption:



7.419.1 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

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

Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

Protected Attributes

- [Real](#) `delta_`
- [Real](#) `deltaForward_`
- [Real](#) `elasticity_`
- [Real](#) `gamma_`
- [Real](#) `theta_`
- [Real](#) `thetaPerDay_`
- [Real](#) `vega_`
- [Real](#) `rho_`
- [Real](#) `dividendRho_`
- [Real](#) `itmCashProbability_`
- `boost::shared_ptr< StochasticProcess > stochasticProcess_`

Classes

- class [arguments](#)
Arguments for single-asset option calculation
- class [results](#)
Results from single-asset option calculation

7.419.2 Member Function Documentation

- 7.419.2.1** [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.) 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.

7.419.2.2 `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](#), [ConvertibleBond::option](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [OneAssetStrikedOption](#), [QuantoForwardVanillaOption](#), and [QuantoVanillaOption](#).

7.419.2.3 void setupExpired () const [protected, virtual]

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

Reimplemented from [Instrument](#).

Reimplemented in [QuantoVanillaOption](#).

7.419.2.4 void performCalculations () const [protected, virtual]

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

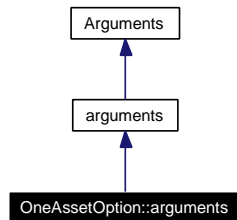
Reimplemented from [Instrument](#).

Reimplemented in [BarrierOption](#), [ForwardVanillaOption](#), [OneAssetStrikedOption](#), and [QuantoVanillaOption](#).

7.420 OneAssetOption::arguments Class Reference

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

Inheritance diagram for OneAssetOption::arguments:



7.420.1 Detailed Description

Arguments for single-asset option calculation

Public Member Functions

- void **validate** () const

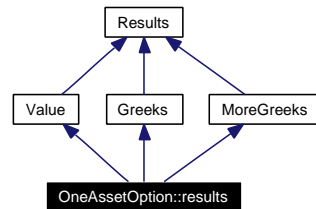
Public Attributes

- boost::shared_ptr< [StochasticProcess](#) > **stochasticProcess**

7.421 OneAssetOption::results Class Reference

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

Inheritance diagram for OneAssetOption::results:



7.421.1 Detailed Description

Results from single-asset option calculation

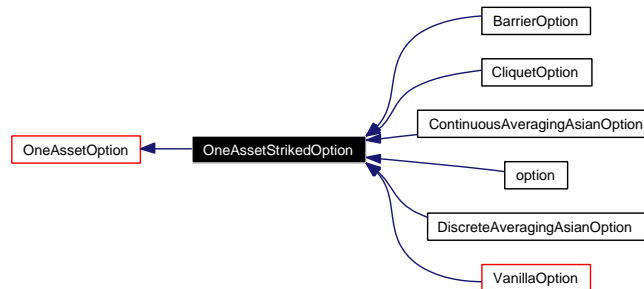
Public Member Functions

- void **reset** ()

7.422 OneAssetStrikedOption Class Reference

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

Inheritance diagram for OneAssetStrikedOption:



7.422.1 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

greeks

- [Real](#) [strikeSensitivity](#) () const

Protected Member Functions

- void [performCalculations](#) () const

Protected Attributes

- [Real](#) [strikeSensitivity_](#)

7.422.2 Member Function Documentation

7.422.2.1 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](#), [ConvertibleBond::option](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [QuantoForwardVanillaOption](#), and [QuantoVanillaOption](#).

7.422.2.2 void performCalculations () const [protected, virtual]

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

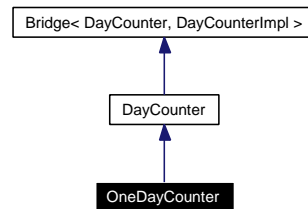
Reimplemented from [OneAssetOption](#).

Reimplemented in [BarrierOption](#), [ForwardVanillaOption](#), and [QuantoVanillaOption](#).

7.423 OneDayCounter Class Reference

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

Inheritance diagram for OneDayCounter:



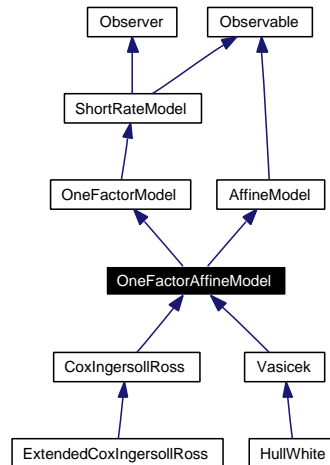
7.423.1 Detailed Description

1/1 day count convention

7.424 OneFactorAffineModel Class Reference

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

Inheritance diagram for OneFactorAffineModel:



7.424.1 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)
- **Real discountBond** ([Time](#) now, [Time](#) maturity, [Rate](#) rate) const
- **DiscountFactor discount** ([Time](#) t) const

Implied discount curve.

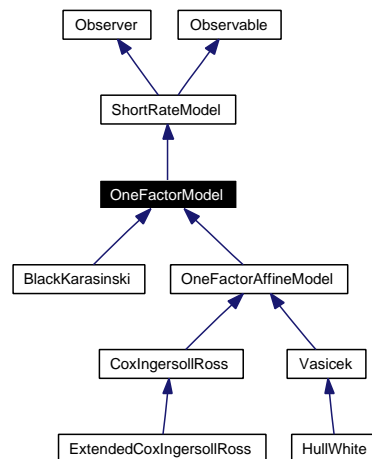
Protected Member Functions

- virtual **Real A** ([Time](#) t, [Time](#) T) const =0
- virtual **Real B** ([Time](#) t, [Time](#) T) const =0

7.425 OneFactorModel Class Reference

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

Inheritance diagram for OneFactorModel:



7.425.1 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.

7.426 OneFactorModel::ShortRateDynamics Class Reference

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

7.426.1 Detailed Description

Base class describing the short-rate dynamics.

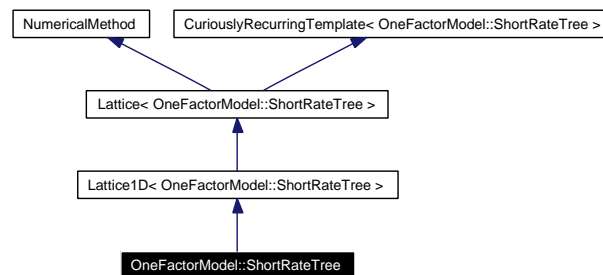
Public Member Functions

- **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.

7.427 OneFactorModel::ShortRateTree Class Reference

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

Inheritance diagram for OneFactorModel::ShortRateTree:



7.427.1 Detailed Description

Recombining trinomial tree discretizing the state variable.

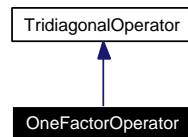
Public Member Functions

- [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

7.428 OneFactorOperator Class Reference

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

Inheritance diagram for OneFactorOperator:



7.428.1 Detailed Description

Interest-rate single factor model differential operator.

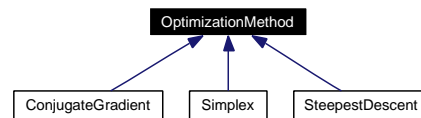
Public Member Functions

- **OneFactorOperator** (const [Array](#) &grid, const boost::shared_ptr< [OneFactorModel::ShortRateDynamics](#) > &)

7.429 OptimizationMethod Class Reference

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

Inheritance diagram for OptimizationMethod:



7.429.1 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

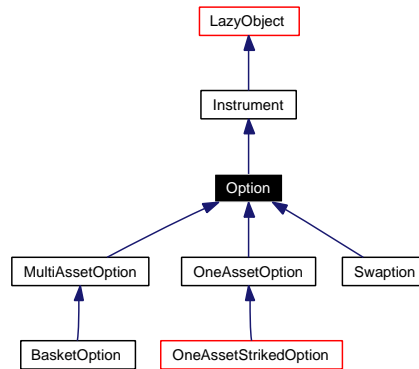
Protected Attributes

- [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_](#)

7.430 Option Class Reference

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

Inheritance diagram for Option:



7.430.1 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)

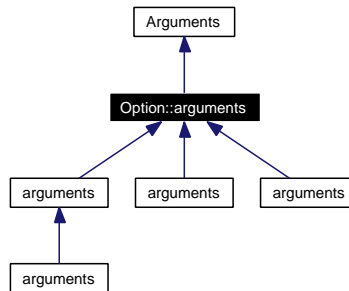
Classes

- class [arguments](#)

7.431 Option::arguments Class Reference

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

Inheritance diagram for Option::arguments:



7.431.1 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

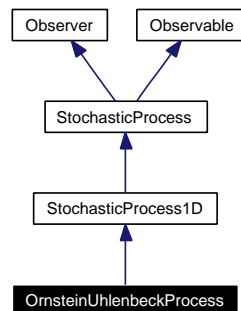
Public Attributes

- boost::shared_ptr< [Payoff](#) > **payoff**
- boost::shared_ptr< [Exercise](#) > **exercise**
- std::vector< [Time](#) > **stoppingTimes**

7.432 OrnsteinUhlenbeckProcess Class Reference

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

Inheritance diagram for OrnsteinUhlenbeckProcess:



7.432.1 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

7.432.2 Member Function Documentation

7.432.2.1 [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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

7.432.2.2 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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

7.432.2.3 Real variance (Time t0, Real x0, Time dt) const [virtual]

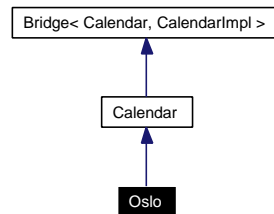
returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess1D](#).

7.433 Oslo Class Reference

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

Inheritance diagram for Oslo:



7.433.1 Detailed Description

Oslo calendar

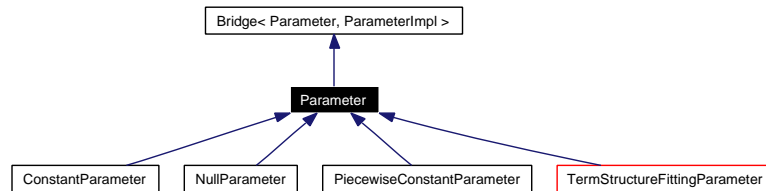
Holidays:

- Saturdays
- Sundays
- Holy Thursday
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- May Day, May 1st
- National Independence Day, May 17st
- Christmas, December 25th
- Boxing Day, December 26th

7.434 Parameter Class Reference

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

Inheritance diagram for Parameter:



7.434.1 Detailed Description

Base class for model arguments.

Public Member Functions

- const [Array](#) & **params** () const
- void **setParam** ([Size](#) i, [Real](#) x)
- bool **testParams** (const [Array](#) ¶ms) 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_**

7.435 ParameterImpl Class Reference

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

7.435.1 Detailed Description

Base class for model parameter implementation.

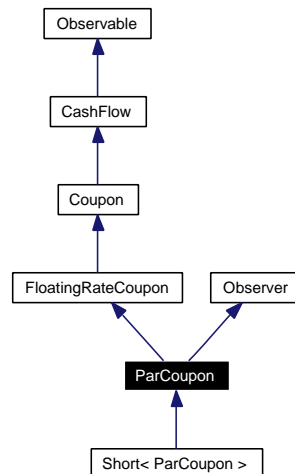
Public Member Functions

- virtual [Real](#) value (const [Array](#) ¶ms, [Time](#) t) const =0

7.436 ParCoupon Class Reference

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

Inheritance diagram for ParCoupon:



7.436.1 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](#) &)

7.436.2 Member Function Documentation

7.436.2.1 [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 >](#).

7.436.2.2 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](#).

7.437 Path Class Reference

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

7.437.1 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

7.438 PathGenerator Class Template Reference

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

7.438.1 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

Examples:

[DiscreteHedging.cpp](#).

Public Types

- typedef [Sample< Path >](#) **sample_type**

Public Member Functions

- **PathGenerator** (const boost::shared_ptr< [StochasticProcess1D](#) > &, [Time](#) length, [Size](#) timeSteps, const GSG &generator, bool brownianBridge)
- **PathGenerator** (const boost::shared_ptr< [StochasticProcess1D](#) > &, 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

7.439 PathPricer Class Template Reference

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

7.439.1 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](#).

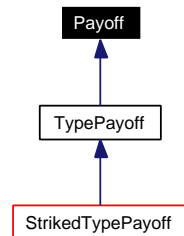
Public Member Functions

- virtual ValueType **operator()** (const PathType &path) const =0

7.440 Payoff Class Reference

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

Inheritance diagram for Payoff:



7.440.1 Detailed Description

Base class for option payoffs.

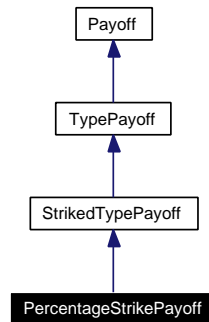
Public Member Functions

- virtual [Real](#) **operator()** ([Real](#) price) const =0

7.441 PercentageStrikePayoff Class Reference

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

Inheritance diagram for PercentageStrikePayoff:



7.441.1 Detailed Description

Payoff with strike expressed as percentage

Public Member Functions

- **PercentageStrikePayoff** (Option::Type type, [Real](#) moneyness)
- **[Real](#) operator()** ([Real](#) price) const

7.442 Period Class Reference

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

7.442.1 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

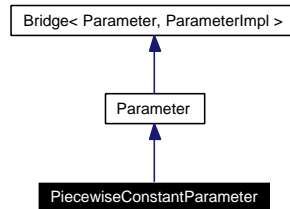
(Note that these are not member functions.)

- [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](#) &)

7.443 PiecewiseConstantParameter Class Reference

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

Inheritance diagram for PiecewiseConstantParameter:



7.443.1 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

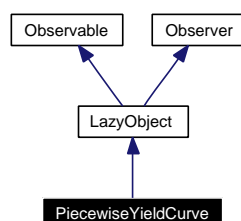
Public Member Functions

- **PiecewiseConstantParameter** (const std::vector< [Time](#) > ×)

7.444 PiecewiseYieldCurve Class Template Reference

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

Inheritance diagram for PiecewiseYieldCurve:



7.444.1 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**

7.444.2 Member Function Documentation

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

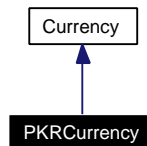
This method must be implemented in derived classes. An instance of **Observer** does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [LazyObject](#).

7.445 PKRCurrency Class Reference

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

Inheritance diagram for PKRCurrency:



7.445.1 Detailed Description

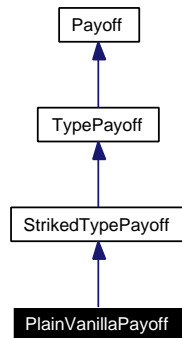
Pakistani rupee.

The ISO three-letter code is PKR; the numeric code is 586. It is divided in 100 paisa.

7.446 PlainVanillaPayoff Class Reference

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

Inheritance diagram for PlainVanillaPayoff:



7.446.1 Detailed Description

Plain-vanilla payoff.

Examples:

[AmericanOption.cpp](#), [DiscreteHedging.cpp](#), and [EuropeanOption.cpp](#).

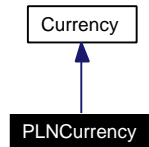
Public Member Functions

- **PlainVanillaPayoff** (Option::Type type, [Real](#) strike)
- [Real](#) **operator()** ([Real](#) price) const

7.447 PLNCurrency Class Reference

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

Inheritance diagram for PLNCurrency:



7.447.1 Detailed Description

Polish zloty.

The ISO three-letter code is PLN; the numeric code is 985. It is divided in 100 groszy.

7.448 PoissonDistribution Class Reference

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

7.448.1 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.

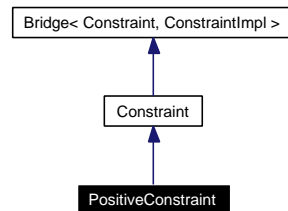
Public Member Functions

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

7.449 PositiveConstraint Class Reference

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

Inheritance diagram for PositiveConstraint:



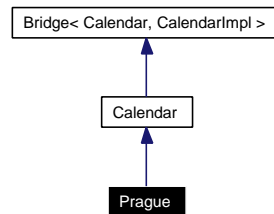
7.449.1 Detailed Description

Constraint imposing positivity to all arguments

7.450 Prague Class Reference

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

Inheritance diagram for Prague:



7.450.1 Detailed Description

Prague calendar

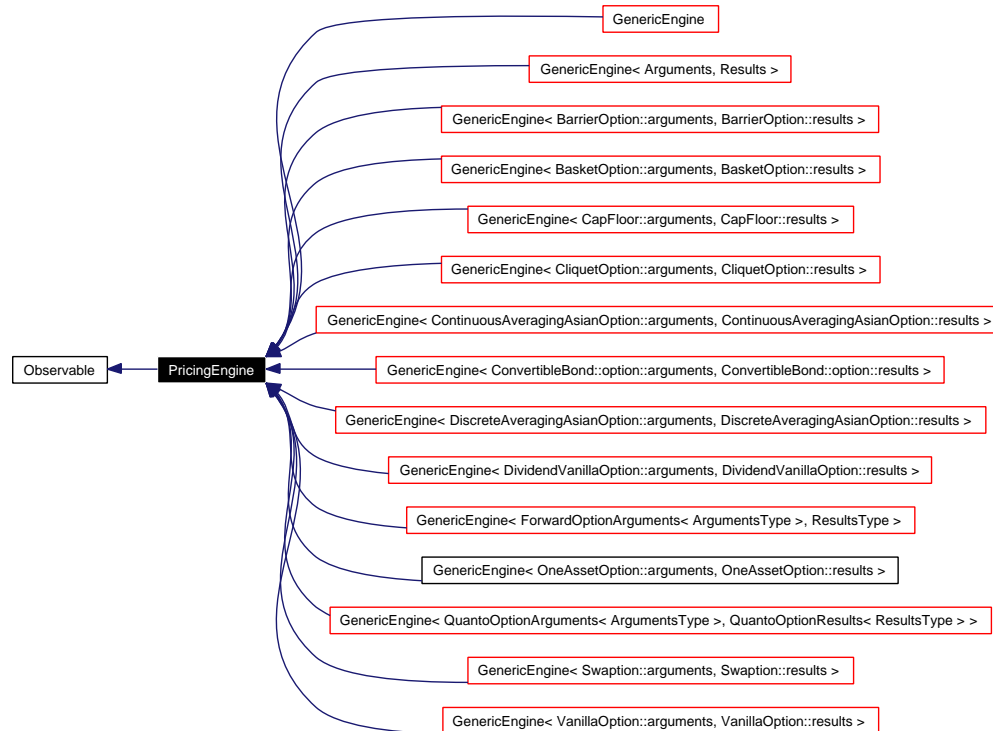
Holidays (see <http://www.pse.cz/>):

- 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

7.451 PricingEngine Class Reference

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

Inheritance diagram for PricingEngine:



7.451.1 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

7.452 PrimeNumbers Class Reference

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

7.452.1 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.

7.453 Problem Class Reference

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

7.453.1 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
- 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

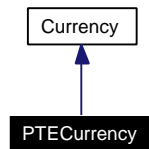
Protected Attributes

- [CostFunction](#) & [costFunction_](#)
Unconstrained cost function.
- [Constraint](#) & [constraint_](#)
Constraint.
- [OptimizationMethod](#) & [method_](#)
constrained optimization method

7.454 PTECurrency Class Reference

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

Inheritance diagram for PTECurrency:



7.454.1 Detailed Description

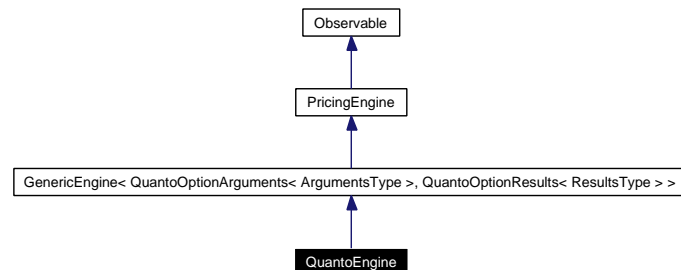
Portuguese escudo.

The ISO three-letter code is PTE; the numeric code is 620. It is divided in 100 centavos.

7.455 QuantoEngine Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

Inheritance diagram for QuantoEngine:



7.455.1 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_**

7.455.2 Member Function Documentation

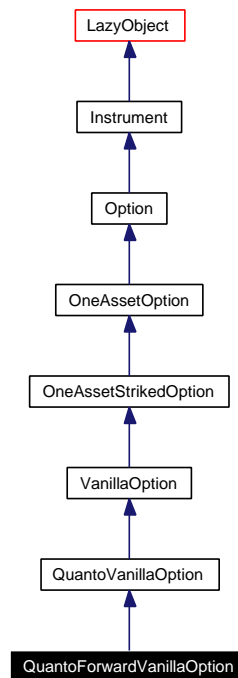
7.455.2.1 ArgumentsType* underlyingArgs () const

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.

7.456 QuantoForwardVanillaOption Class Reference

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

Inheritance diagram for QuantoForwardVanillaOption:



7.456.1 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

7.456.2 Member Function Documentation

7.456.2.1 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](#).

7.457 QuantoOptionArguments Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

7.457.1 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`

7.458 QuantoOptionResults Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

7.458.1 Detailed Description

```
template<class ResultsType> class QuantLib::QuantoOptionResults< ResultsType >
```

Results from quanto option calculation

Public Member Functions

- void reset ()

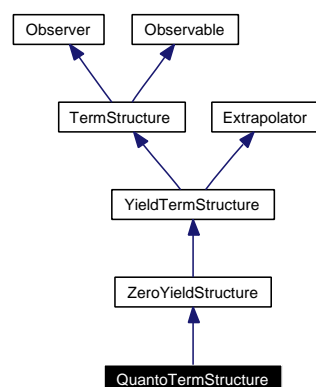
Public Attributes

- [Real](#) qvega
- [Real](#) qrho
- [Real](#) qlambda

7.459 QuantoTermStructure Class Reference

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

Inheritance diagram for QuantoTermStructure:



7.459.1 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

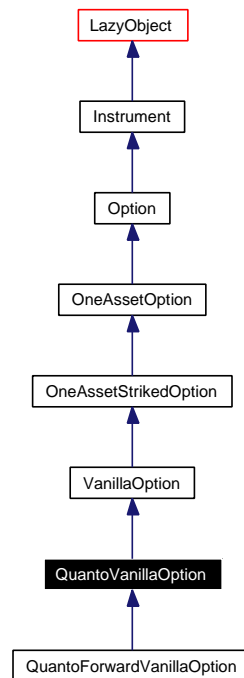
Protected Member Functions

- [Rate zeroYieldImpl](#) ([Time](#)) const
returns the zero yield as seen from the evaluation date

7.460 QuantoVanillaOption Class Reference

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

Inheritance diagram for QuantoVanillaOption:



7.460.1 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

greeks

- [Real](#) [qvega](#) () const

- [Real](#) `qrho` () const
- [Real](#) `qlambda` () const

Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

Protected Attributes

- [Handle](#)< [YieldTermStructure](#) > `foreignRiskFreeTS_`
- [Handle](#)< [BlackVolTermStructure](#) > `exchRateVolTS_`
- [Handle](#)< [Quote](#) > `correlation_`
- [Real](#) `qvega_`
- [Real](#) `qrho_`
- [Real](#) `qlambda_`

7.460.2 Member Function Documentation

7.460.2.1 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](#).

7.460.2.2 void [setupExpired](#) () const [protected, virtual]

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

Reimplemented from [OneAssetOption](#).

7.460.2.3 void [performCalculations](#) () const [protected, virtual]

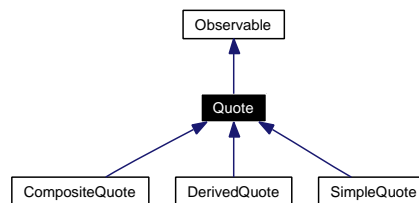
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [OneAssetStrikedOption](#).

7.461 Quote Class Reference

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

Inheritance diagram for Quote:



7.461.1 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

7.462 RamdomizedLDS Class Template Reference

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

7.462.1 Detailed Description

```
template<class LDS, class PRS = RandomSequenceGenerator<MersenneTwisterUniform-
Rng>> class QuantLib::RamdomizedLDS< 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

- **RamdomizedLDS** (const LDS &lds, const PRS &prsg)
- **RamdomizedLDS** (const LDS &lds)
- **RamdomizedLDS** ([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

7.462.2 Member Function Documentation

7.462.2.1 void nextRandomizer ()

update the randomizing vector and re-initialize the low discrepancy generator

7.463 RandomSequenceGenerator Class Template Reference

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

7.463.1 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`

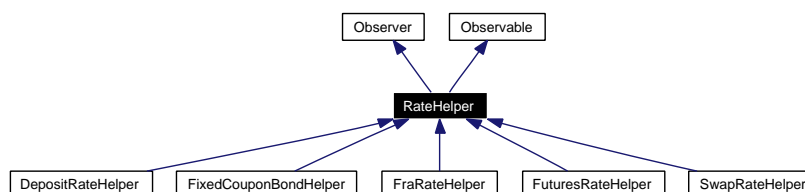
Public Member Functions

- [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

7.464 RateHelper Class Reference

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

Inheritance diagram for RateHelper:



7.464.1 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](#) > "e)
- [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_](#)

7.464.2 Member Function Documentation

7.464.2.1 `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](#).

7.464.2.2 `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](#).

7.464.2.3 `void update ()` [virtual]

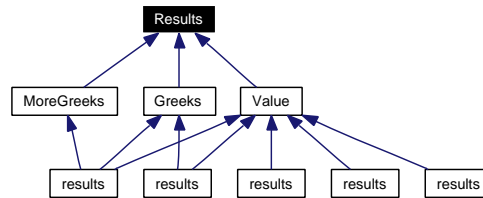
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.465 Results Class Reference

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

Inheritance diagram for Results:



7.465.1 Detailed Description

base class for generic result groups

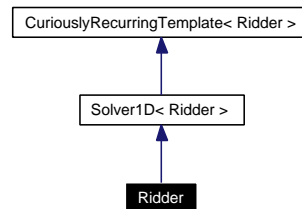
Public Member Functions

- virtual void **reset** ()=0

7.466 Ridder Class Reference

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

Inheritance diagram for Ridder:



7.466.1 Detailed Description

Ridder 1-D solver

Tests

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

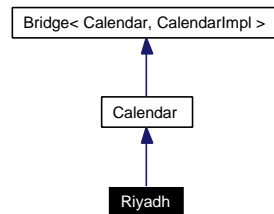
Public Member Functions

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

7.467 Riyadh Class Reference

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

Inheritance diagram for Riyadh:



7.467.1 Detailed Description

Riyadh calendar

Holidays:

- Fridays

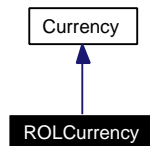
Other holidays for which no rule is given (data available for 2004-2005 only:)

- EID AL-ADHA
- EID AL-FITR

7.468 ROLCurrency Class Reference

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

Inheritance diagram for ROLCurrency:



7.468.1 Detailed Description

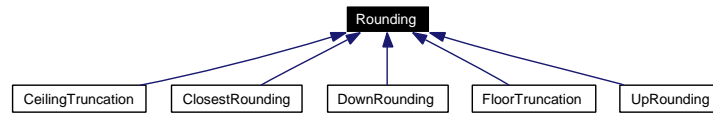
Romanian leu.

The ISO three-letter code is ROL; the numeric code is 642. It is divided in 100 bani.

7.469 Rounding Class Reference

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

Inheritance diagram for Rounding:



7.469.1 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

7.469.2 Member Enumeration Documentation

7.469.2.1 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

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

7.469.3 Constructor & Destructor Documentation

7.469.3.1 [Rounding \(\)](#)

default constructor

Instances built through this constructor don't perform any rounding.

7.470 SalvagingAlgorithm Struct Reference

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

7.470.1 Detailed Description

algorithm used for matricial pseudo square root

Public Types

- enum Type { None, Spectral, Hypersphere }

7.471 Sample Struct Template Reference

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

7.471.1 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

7.472 SampledCurve Class Reference

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

7.472.1 Detailed Description

This class contains a sampled curve.

Initially the class will contain one indexed curve

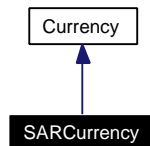
Public Member Functions

- **SampledCurve** ([Size](#) gridSize)
- void **setLogSpacing** ([Real](#) min, [Real](#) max)
- void **setLinearSpacing** ([Real](#) min, [Real](#) max)
- template<class F> void **sample** (F &f)
- void **setGrid** (const [Array](#) &g)
- void **setValues** (const [Array](#) &g)
- [Array](#) & **grid** ()
- [Array](#) & **values** ()
- [Real](#) & **value** (int i)

7.473 SARCurrency Class Reference

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

Inheritance diagram for SARCurrency:



7.473.1 Detailed Description

Saudi riyal.

The ISO three-letter code is SAR; the numeric code is 682. It is divided in 100 halalat.

7.474 Schedule Class Reference

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

7.474.1 Detailed Description

Payment schedule.

Examples:

[BermudanSwaption.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

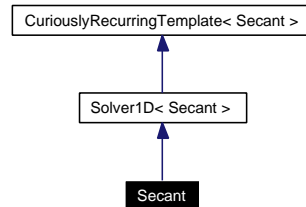
Other inspectors

- const [Calendar](#) & **calendar** () const
- const [Date](#) & **startDate** () const
- const [Date](#) & **endDate** () const
- [Frequency](#) **frequency** () const
- [BusinessDayConvention](#) **businessDayConvention** () const

7.475 Secant Class Reference

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

Inheritance diagram for Secant:



7.475.1 Detailed Description

Secant 1-D solver

Tests

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

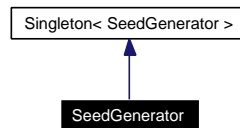
Public Member Functions

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

7.476 SeedGenerator Class Reference

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

Inheritance diagram for SeedGenerator:



7.476.1 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>`

7.477 SegmentIntegral Class Reference

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

7.477.1 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.

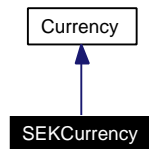
Public Member Functions

- **SegmentIntegral** ([Size](#) intervals)
- **template<class F> Real operator()** (const F &f, [Real](#) a, [Real](#) b) const

7.478 SEKCurrency Class Reference

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

Inheritance diagram for SEKCurrency:



7.478.1 Detailed Description

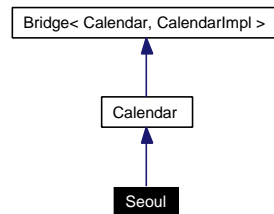
Swedish krona.

The ISO three-letter code is SEK; the numeric code is 752. It is divided in 100 öre.

7.479 Seoul Class Reference

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

Inheritance diagram for Seoul:



7.479.1 Detailed Description

Seoul calendar

Holidays:

- 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

Data from <http://www.kofex.com>

7.480 SequenceStatistics Class Template Reference

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

7.480.1 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)`

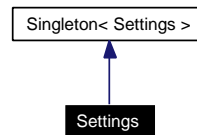
Protected Attributes

- `Size dimension_`
- `std::vector< statistics_type > stats_`
- `std::vector< Real > results_`
- `Matrix quadraticSum_`

7.481 Settings Class Reference

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

Inheritance diagram for Settings:



7.481.1 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 &)

7.481.2 Member Function Documentation

7.481.2.1 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.

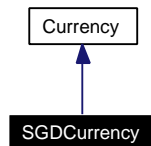
Warning:

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.

7.482 SGDCurrency Class Reference

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

Inheritance diagram for SGDCurrency:



7.482.1 Detailed Description

[Singapore](#) dollar.

The ISO three-letter code is SGD; the numeric code is 702. It is divided in 100 cents.

7.483 Short Class Template Reference

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

7.483.1 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

7.483.2 Member Function Documentation

7.483.2.1 [Real amount](#) () const

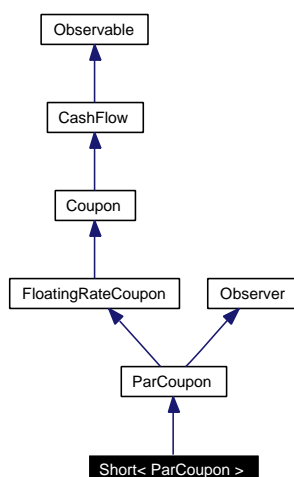
inhibit calculation

Unlike [ParCoupon](#), this coupon can't calculate its fixing for future dates, either.

7.484 Short< ParCoupon > Class Template Reference

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

Inheritance diagram for Short< ParCoupon >:



7.484.1 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

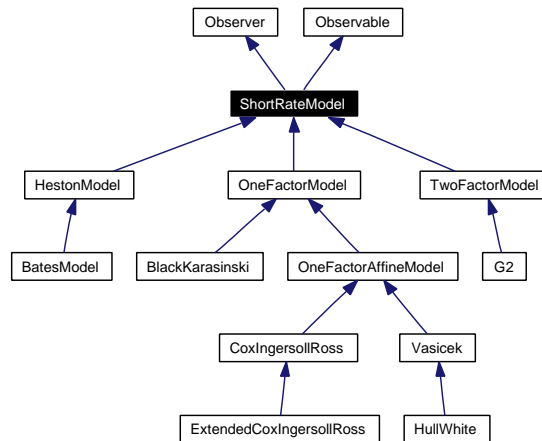
Visitability

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

7.485 ShortRateModel Class Reference

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

Inheritance diagram for ShortRateModel:



7.485.1 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](#)())
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.
- void **setParams** (const [Array](#) ¶ms)

Protected Member Functions

- virtual void **generateArguments** ()

Protected Attributes

- std::vector< [Parameter](#) > **arguments_**
- boost::shared_ptr< [Constraint](#) > **constraint_**

Friends

- class `CalibrationFunction`

7.485.2 Member Function Documentation

7.485.2.1 `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](#).

7.485.2.2 `void calibrate (const std::vector< boost::shared_ptr< CalibrationHelper > > &, OptimizationMethod & method, const Constraint & constraint = Constraint\(\))`

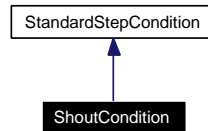
Calibrate to a set of market instruments (caps/swaptions).

An additional constraint can be passed which must be satisfied in addition to the constraints of the model.

7.486 ShoutCondition Class Reference

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

Inheritance diagram for ShoutCondition:



7.486.1 Detailed Description

Shout option condition.

A shout option is an option where the holder has the right to lock in a minimum value for the payoff at one (shout) time during the option's life. The minimum value is the option's intrinsic value at the shout time.

Todo

unify the intrinsicValues/Payoff thing

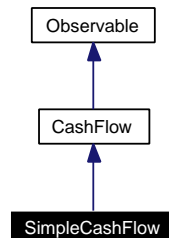
Public Member Functions

- **ShoutCondition** (Option::Type type, [Real](#) strike, [Time](#) resTime, [Rate](#) rate)
- **ShoutCondition** (const [Array](#) &intrinsicValues, [Time](#) resTime, [Rate](#) rate)
- void **applyTo** ([Array](#) &a, [Time](#) t) const

7.487 SimpleCashFlow Class Reference

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

Inheritance diagram for SimpleCashFlow:



7.487.1 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
returns the date at which the cash flow is settled

Visitability

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

7.487.2 Member Function Documentation

7.487.2.1 [Real](#) amount () const [virtual]

returns the amount of the cash flow

Note:

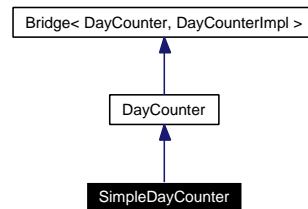
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

7.488 SimpleDayCounter Class Reference

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

Inheritance diagram for SimpleDayCounter:



7.488.1 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.

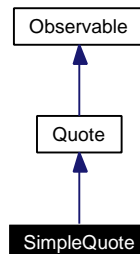
Tests

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

7.489 SimpleQuote Class Reference

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

Inheritance diagram for SimpleQuote:



7.489.1 Detailed Description

market element returning a stored value

Examples:

[AmericanOption.cpp](#), [BermudanSwaption.cpp](#), [DiscreteHedging.cpp](#), [EuropeanOption.cpp](#), and [swapvaluation.cpp](#).

Public Member Functions

- **SimpleQuote** ([Real](#) value)

Quote interface

- [Real](#) value () const
returns the current value

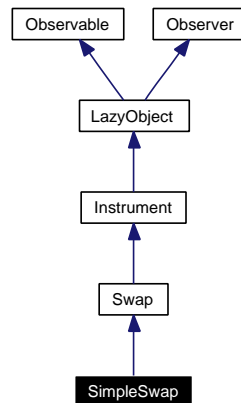
Modifiers

- void **setValue** ([Real](#) value)

7.490 SimpleSwap Class Reference

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

Inheritance diagram for SimpleSwap:



7.490.1 Detailed Description

Simple fixed-rate vs [Libor](#) 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

- **SimpleSwap** (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)
- [Rate](#) **fairRate** () const
- [Spread](#) **fairSpread** () const
- [Real](#) **fixedLegBPS** () const
- [Real](#) **floatingLegBPS** () const
- [Rate](#) **fixedRate** () const

- [Spread](#) **spread** () const
- [Real](#) **nominal** () const
- **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

Classes

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

7.490.2 Member Function Documentation

7.490.2.1 void setupArguments ([Arguments](#) * args) const [virtual]

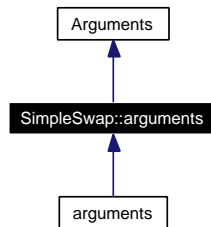
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

7.491 SimpleSwap::arguments Class Reference

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

Inheritance diagram for SimpleSwap::arguments:



7.491.1 Detailed Description

Arguments for simple swap calculation

Public Member Functions

- void **validate** () const

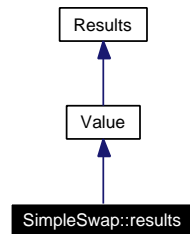
Public Attributes

- 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**

7.492 SimpleSwap::results Class Reference

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

Inheritance diagram for SimpleSwap::results:



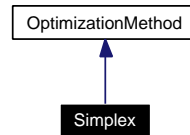
7.492.1 Detailed Description

Results from simple swap calculation

7.493 Simplex Class Reference

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

Inheritance diagram for Simplex:



7.493.1 Detailed Description

Multi-dimensional simplex class.

Examples:

[BermudanSwaption.cpp](#).

Public Member Functions

- [Simplex](#) ([Real](#) lambda, [Real](#) tol)
- virtual void [minimize](#) (const [Problem](#) &P) const
minimize the optimization problem P

7.493.2 Constructor & Destructor Documentation

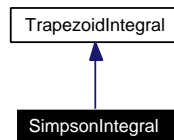
7.493.2.1 [Simplex](#) ([Real](#) lambda, [Real](#) tol)

Constructor taking as input the characteristic length and tolerance

7.494 SimpsonIntegral Class Reference

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

Inheritance diagram for SimpsonIntegral:



7.494.1 Detailed Description

Integral of a one-dimensional function.

Tests

the correctness of the result is tested by checking it against known good values.

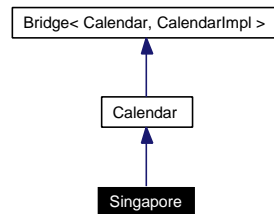
Public Member Functions

- **SimpsonIntegral** ([Real](#) accuracy, [Size](#) maxIterations=[Null](#)< [Size](#) >())
- **template<class F> [Real](#) operator()** (const F &f, [Real](#) a, [Real](#) b) const

7.495 Singapore Class Reference

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

Inheritance diagram for Singapore:



7.495.1 Detailed Description

Singapore calendar

Holidays:

- 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

Data from <http://www.asx.com.au> and <http://www.ses.com.sg>

7.496 SingleAsset Struct Template Reference

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

7.496.1 Detailed Description

```
template<class rng_traits = PseudoRandom> struct QuantLib::SingleAsset< rng_traits >
```

Deprecated

use [SingleVariate](#) instead

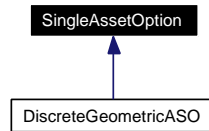
Public Types

- typedef [SingleVariate](#)< rng_traits >::path_type path_type
- typedef [SingleVariate](#)< rng_traits >::path_pricer_type path_pricer_type
- typedef [SingleVariate](#)< rng_traits >::rsg_type rsg_type
- typedef [SingleVariate](#)< rng_traits >::path_generator_type path_generator_type

7.497 SingleAssetOption Class Reference

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

Inheritance diagram for SingleAssetOption:



7.497.1 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

7.497.2 Member Function Documentation

7.497.2.1 [Volatility](#) impliedVolatility ([Real](#) targetValue, [Real](#) accuracy = 1e-4, [Size](#) maxEvaluations = 100, [Volatility](#) minVol = QL_MIN_VOLATILITY, [Volatility](#) maxVol = QL_MAX_VOLATILITY) const

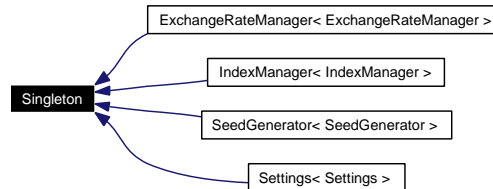
Warning:

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.

7.498 Singleton Class Template Reference

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

Inheritance diagram for Singleton:



7.498.1 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

7.499 SingleVariate Struct Template Reference

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

7.499.1 Detailed Description

```
template<class rng_traits = PseudoRandom> struct QuantLib::SingleVariate< rng_traits >
```

default Monte Carlo traits for single-variate models

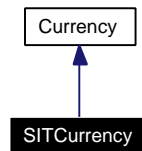
Public Types

- typedef [Path](#) **path_type**
- typedef [PathPricer](#)< [path_type](#) > **path_pricer_type**
- typedef rng_traits::rsg_type **rsg_type**
- typedef [PathGenerator](#)< rsg_type > **path_generator_type**

7.500 SITCurrency Class Reference

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

Inheritance diagram for SITCurrency:



7.500.1 Detailed Description

Slovenian tolar.

The ISO three-letter code is SIT; the numeric code is 705. It is divided in 100 stotinov.

7.501 SKKCurrency Class Reference

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

Inheritance diagram for SKKCurrency:



7.501.1 Detailed Description

Slovak koruna.

The ISO three-letter code is SKK; the numeric code is 703. It is divided in 100 halierov.

7.502 SobolRsg Class Reference

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

7.502.1 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

7.502.2 Constructor & Destructor Documentation

- 7.502.2.1 `SobolRsg` (`Size` dimensionality, unsigned long seed = 0, `DirectionIntegers` directionIntegers = `Jaeckel`)

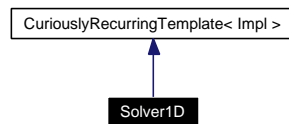
Precondition:

dimensionality must be <= PPMT_MAX_DIM

7.503 Solver1D Class Template Reference

```
#include <ql/solver1d.hpp>
```

Inheritance diagram for Solver1D:



7.503.1 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_

7.503.2 Member Function Documentation

7.503.2.1 [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.

7.503.2.2 [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).

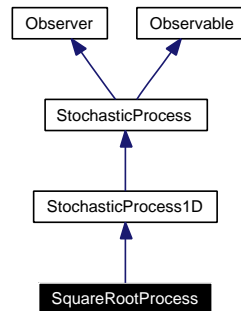
7.503.2.3 void setMaxEvaluations ([Size](#) *evaluations*)

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.

7.504 SquareRootProcess Class Reference

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

Inheritance diagram for SquareRootProcess:



7.504.1 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. $\sigma(t, x_t)$

7.505 StatsHolder Class Reference

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

7.505.1 Detailed Description

Helper class for precomputed distributions.

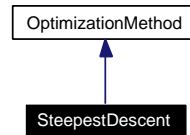
Public Member Functions

- **StatsHolder** ([Real](#) mean, [Real](#) standardDeviation)
- [Real](#) **mean** () const
- [Real](#) **standardDeviation** () const

7.506 SteepestDescent Class Reference

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

Inheritance diagram for SteepestDescent:



7.506.1 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

7.507 `step_iterator` Class Template Reference

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

7.507.1 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

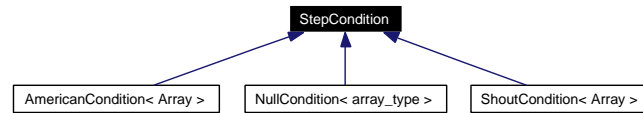
(Note that these are not member functions.)

- [step_iterator](#)< Iterator > [make_step_iterator](#) (Iterator it, [Size](#) step)
helper function to create step iterators

7.508 StepCondition Class Template Reference

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

Inheritance diagram for StepCondition:



7.508.1 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

7.509 StepConditionSet Class Template Reference

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

7.509.1 Detailed Description

```
template<typename array_type> class QuantLib::StepConditionSet< array_type >
```

Parallel evolver for multiple arrays.

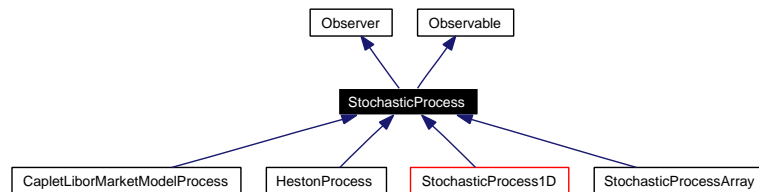
Public Member Functions

- void **applyTo** (std::vector< array_type > &a, [Time](#) t) const
- void **push_back** (const itemType &a)

7.510 StochasticProcess Class Reference

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

Inheritance diagram for StochasticProcess:



7.510.1 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

7.510.2 Member Function Documentation

7.510.2.1 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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [StochasticProcessArray](#).

7.510.2.2 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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [StochasticProcessArray](#).

7.510.2.3 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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [StochasticProcessArray](#).

7.510.2.4 virtual [Disposable](#)<[Array](#)> **evolve** ([Time](#) *t0*, const [Array](#) & *x0*, [Time](#) *dt*, const [Array](#) & *dw*) const [virtual]

returns the asset value after a time interval Δt 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 [CapletLiborMarketModelProcess](#).

7.510.2.5 `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 [CapletLiborMarketModelProcess](#), [HestonProcess](#), and [StochasticProcess-Array](#).

7.510.2.6 `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 [StochasticProcess-Array](#).

7.510.2.7 `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.

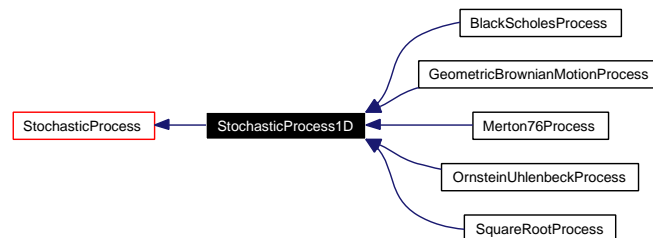
Implements [Observer](#).

Reimplemented in [BlackScholesProcess](#).

7.511 StochasticProcess1D Class Reference

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

Inheritance diagram for StochasticProcess1D:



7.511.1 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

7.511.2 Member Function Documentation

7.511.2.1 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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [OrnsteinUhlenbeckProcess](#).

7.511.2.2 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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [OrnsteinUhlenbeckProcess](#).

7.511.2.3 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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [OrnsteinUhlenbeckProcess](#).

7.511.2.4 virtual [Real](#) evolve ([Time](#) *t0*, [Real](#) *x0*, [Time](#) *dt*, [Real](#) *dw*) const [virtual]

returns the asset value after a time interval Δt 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.

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

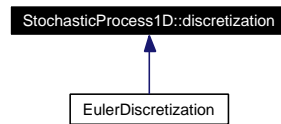
applies a change to the asset value. By default, it returns $x + \Delta x$.

Reimplemented in [BlackScholesProcess](#), and [Merton76Process](#).

7.512 StochasticProcess1D::discretization Class Reference

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

Inheritance diagram for StochasticProcess1D::discretization:



7.512.1 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

7.513 StochasticProcess::discretization Class Reference

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

Inheritance diagram for StochasticProcess::discretization:



7.513.1 Detailed Description

discretization of a stochastic process over a given time interval

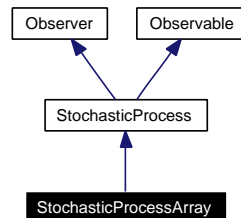
Public Member Functions

- 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

7.514 StochasticProcessArray Class Reference

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

Inheritance diagram for StochasticProcessArray:



7.514.1 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_**

7.514.2 Member Function Documentation

7.514.2.1 `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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

7.514.2.2 `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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

7.514.2.3 `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 Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

7.514.2.4 `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](#).

7.514.2.5 `Time time (const Date &) const` [virtual]

returns the time value corresponding to the given date in the reference system of the stochastic process.

Note:

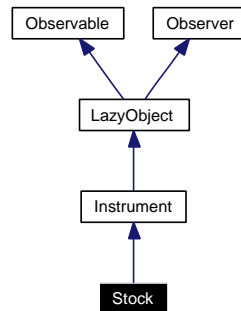
As a number of processes might not need this functionality, a default implementation is given which raises an exception.

Reimplemented from [StochasticProcess](#).

7.515 Stock Class Reference

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

Inheritance diagram for Stock:



7.515.1 Detailed Description

Simple stock class.

Public Member Functions

- **Stock** (const [Handle](#)< [Quote](#) > "e)
- bool [isExpired](#) () const
returns whether the instrument is still tradable.

Protected Member Functions

- void [performCalculations](#) () const

7.515.2 Member Function Documentation

7.515.2.1 void performCalculations () const [protected, virtual]

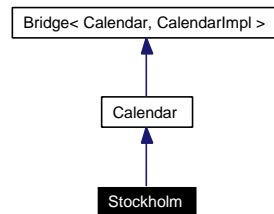
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

7.516 Stockholm Class Reference

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

Inheritance diagram for Stockholm:



7.516.1 Detailed Description

Stockholm calendar

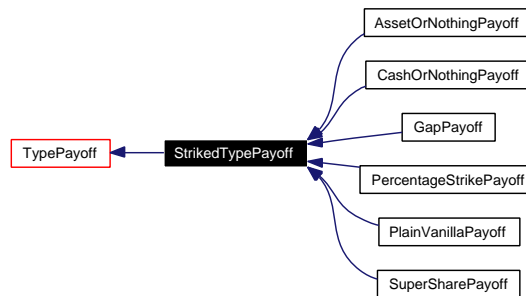
Holidays:

- Saturdays
- Sundays
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- Midsummer Eve (Friday between June 18-24)
- New Year's Day, January 1st
- Epiphany, January 6th
- May Day, May 1st
- National Day, June 6th
- Christmas Eve, December 24th
- Christmas Day, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31th

7.517 StrikedTypePayoff Class Reference

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

Inheritance diagram for StrikedTypePayoff:



7.517.1 Detailed Description

Intermediate class for payoffs based on a fixed strike.

Public Member Functions

- `StrikedTypePayoff` (Option::Type type, Real strike)
- `Real strike` () const

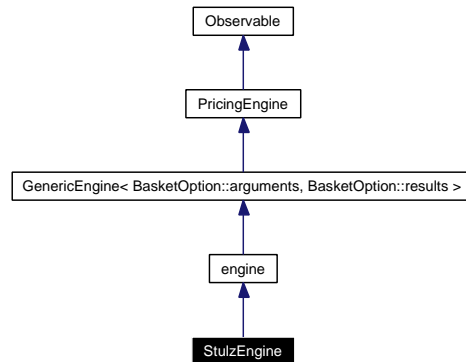
Protected Attributes

- `Real strike_`

7.518 StulzEngine Class Reference

```
#include <ql/PricingEngines/Basket/stulzengine.hpp>
```

Inheritance diagram for StulzEngine:



7.518.1 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.

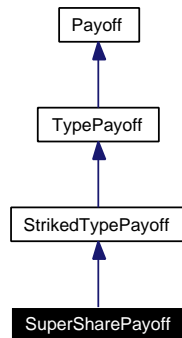
Public Member Functions

- void **calculate** () const

7.519 SuperSharePayoff Class Reference

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

Inheritance diagram for SuperSharePayoff:



7.519.1 Detailed Description

Binary supershare payoff.

Public Member Functions

- **SuperSharePayoff** (Option::Type type, [Real](#) strike, [Real](#) strikeIncrement)
- **[Real](#) operator()** ([Real](#) price) const
- **[Real](#) strikeIncrement** () const

7.520 SVD Class Reference

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

7.520.1 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.

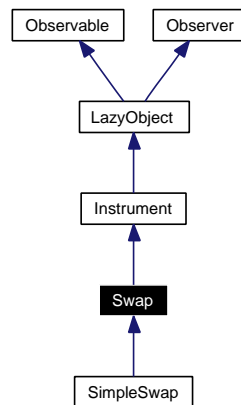
Public Member Functions

- **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** ()

7.521 Swap Class Reference

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

Inheritance diagram for Swap:



7.521.1 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_`

7.521.2 Member Function Documentation

7.521.2.1 [TimeBasket](#) sensitivity ([Integer](#) *basis* = 2) const

[Bug](#)

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

7.521.2.2 `void setupExpired () const` [protected, virtual]

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

Reimplemented from [Instrument](#).

7.521.2.3 `void performCalculations () const` [protected, virtual]

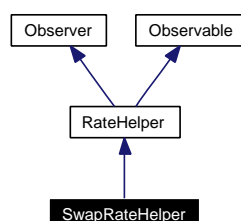
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

7.522 SwapRateHelper Class Reference

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

Inheritance diagram for SwapRateHelper:



7.522.1 Detailed Description

Swap rate

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)
- **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_`
- [Date](#) `settlement_`
- [Date](#) `latestDate_`
- `boost::shared_ptr< SimpleSwap > swap_`
- `Handle< YieldTermStructure > termStructureHandle_`

7.522.2 Member Function Documentation

7.522.2.1 [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](#).

7.522.2.2 `void setTermStructure (YieldTermStructure *)` [virtual]

sets the term structure to be used for pricing

Warning:

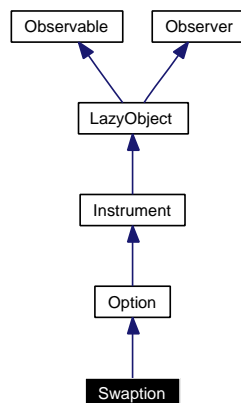
Being a pointer and not a `shared_ptr`, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

7.523 Swaption Class Reference

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

Inheritance diagram for Swaption:



7.523.1 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< [SimpleSwap](#) > &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

7.523.2 Member Function Documentation

7.523.2.1 void setupArguments ([Arguments](#) *) const [virtual]

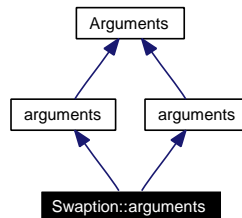
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

7.524 Swaption::arguments Class Reference

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

Inheritance diagram for Swaption::arguments:



7.524.1 Detailed Description

Arguments for swaption calculation

Public Member Functions

- `void validate () const`

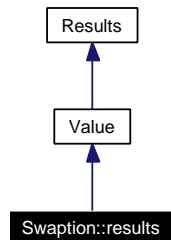
Public Attributes

- [Rate](#) `fairRate`
- [Rate](#) `fixedRate`
- [Real](#) `fixedBPS`

7.525 Swaption::results Class Reference

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

Inheritance diagram for Swaption::results:



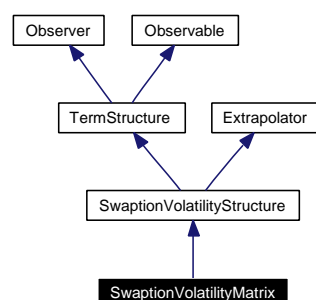
7.525.1 Detailed Description

Results from swaption calculation

7.526 SwaptionVolatilityMatrix Class Reference

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

Inheritance diagram for SwaptionVolatilityMatrix:



7.526.1 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

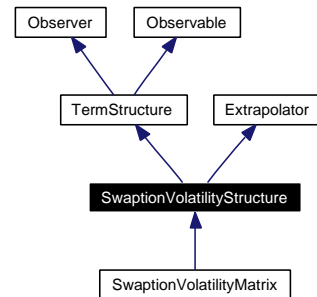
- **SwaptionVolatilityMatrix** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &exerciseDates, const std::vector< [Period](#) > &lengths, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter)
- [DayCounter](#) dayCounter () const
the day counter used for date/time conversion
- 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

7.527 SwaptionVolatilityStructure Class Reference

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

Inheritance diagram for SwaptionVolatilityStructure:



7.527.1 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

7.527.2 Constructor & Destructor Documentation

7.527.2.1 [SwaptionVolatilityStructure](#) ()

default constructor

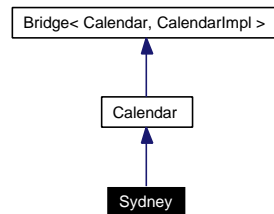
Warning:

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

7.528 Sydney Class Reference

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

Inheritance diagram for Sydney:



7.528.1 Detailed Description

Sydney calendar (New South Wales, Australia)

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Australia Day, January 26th (possibly moved to Monday)
- Good Friday
- Easter Monday
- ANZAC Day, April 25th (possibly moved to Monday)
- Queen's Birthday, second Monday in June
- Bank Holiday, first Monday in August
- Labour Day, first Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

7.529 SymmetricSchurDecomposition Class Reference

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

7.529.1 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

7.529.2 Constructor & Destructor Documentation

7.529.2.1 [SymmetricSchurDecomposition](#) (const [Matrix](#) & s)

Precondition:

s must be symmetric

7.530 TabulatedGaussLegendre Class Reference

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

7.530.1 Detailed Description

tabulated Gauss-Legendre quadratures

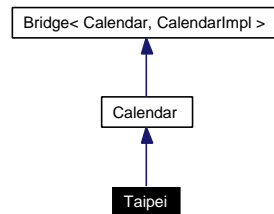
Public Member Functions

- **TabulatedGaussLegendre** ([Size](#) n=20)
- **template<class F> [Real](#) operator()** (const F &f) const
- **void order** ([Size](#))
- **[Size](#) order** () const

7.531 Taipei Class Reference

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

Inheritance diagram for Taipei:



7.531.1 Detailed Description

Taipei calendar

Holidays (data from <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

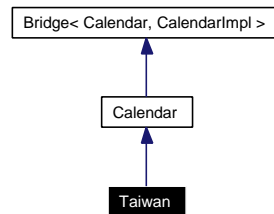
Other holidays for which no rule is given (data available for 2002-2005 only:)

- Chinese Lunar New Year
- Tomb Sweeping Day
- Dragon Boat Festival
- Moon Festival

7.532 Taiwan Class Reference

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

Inheritance diagram for Taiwan:



7.532.1 Detailed Description

Taiwan calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Peace Day, February 28th
- Labor Day, May 1st
- National Day, October 10th

Other holidays for which no rule is given (data available for 2004-2006 only:)

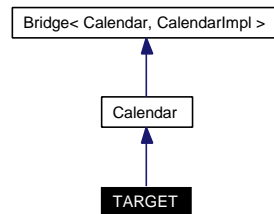
- Lunar New Year
- Tomb Sweeping Day
- Dragon Boat Day
- Mid-Autumn Festival

Data from <http://www.taifex.com.tw>

7.533 TARGET Class Reference

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

Inheritance diagram for TARGET:



7.533.1 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.

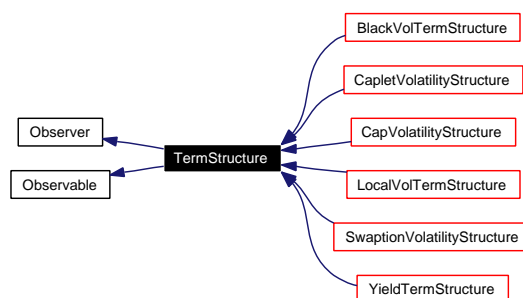
Examples:

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

7.534 TermStructure Class Reference

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

Inheritance diagram for TermStructure:



7.534.1 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

7.534.2 Constructor & Destructor Documentation

7.534.2.1 [TermStructure](#) ()

default constructor

Warning:

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

7.534.3 Member Function Documentation

7.534.3.1 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.

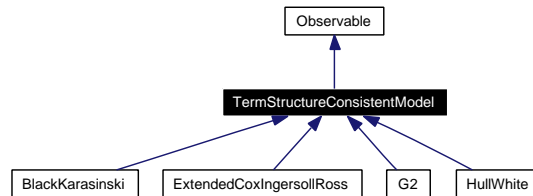
Implements [Observer](#).

Reimplemented in [AffineTermStructure](#), [ExtendedDiscountCurve](#), [FlatForward](#), and [CapVolatilityVector](#).

7.535 TermStructureConsistentModel Class Reference

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

Inheritance diagram for TermStructureConsistentModel:



7.535.1 Detailed Description

Term-structure consistent model class.

This is a base class for models that can reprice exactly any discount bond.

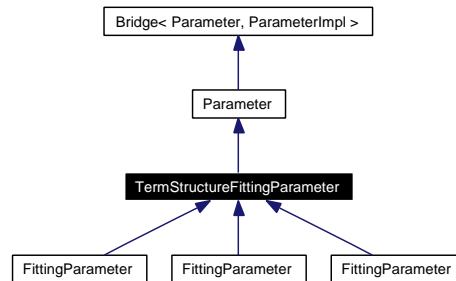
Public Member Functions

- `TermStructureConsistentModel` (const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- const [Handle](#)< [YieldTermStructure](#) > & `termStructure` () const

7.536 TermStructureFittingParameter Class Reference

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

Inheritance diagram for TermStructureFittingParameter:



7.536.1 Detailed Description

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

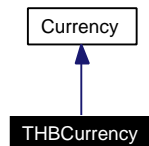
Public Member Functions

- `TermStructureFittingParameter` (const boost::shared_ptr< Parameter::Impl > &impl)
- `TermStructureFittingParameter` (const [Handle](#)< [YieldTermStructure](#) > &term)

7.537 THBCurrency Class Reference

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

Inheritance diagram for THBCurrency:



7.537.1 Detailed Description

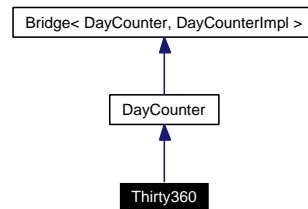
Thai baht.

The ISO three-letter code is THB; the numeric code is 764. It is divided in 100 stang.

7.538 Thirty360 Class Reference

```
#include <ql/DayCounters/thirty360.hpp>
```

Inheritance diagram for Thirty360:



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

Public Types

- enum **Convention** {
 USA, **BondBasis**, **European**, **EurobondBasis**,
 Italian }

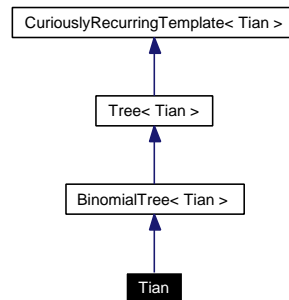
Public Member Functions

- **Thirty360** (Convention c=Thirty360::USA)

7.539 Tian Class Reference

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

Inheritance diagram for Tian:



7.539.1 Detailed Description

Tian tree: third moment matching, multiplicative approach

Public Member Functions

- **Tian** (const boost::shared_ptr< [StochasticProcess1D](#) > &process, [Time](#) end, [Size](#) steps, [Real](#) strike)
- [Real](#) underlying ([Size](#) i, [Size](#) index) const
- [Real](#) probability ([Size](#), [Size](#), [Size](#) branch) const

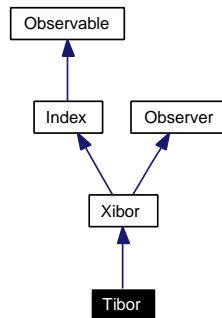
Protected Attributes

- [Real](#) up_
- [Real](#) down_
- [Real](#) pu_
- [Real](#) pd_

7.540 Tibor Class Reference

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

Inheritance diagram for Tibor:



7.540.1 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](#)())

7.541 TimeBasket Class Reference

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

7.541.1 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

7.542 TimeGrid Class Reference

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

7.542.1 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
- const std::vector< [Time](#) > & **mandatoryTimes** () const
- [Time](#) **dt** ([Size](#) i) const

7.542.2 Constructor & Destructor Documentation

7.542.2.1 **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.

7.542.2.2 **TimeGrid** (Iterator *begin*, Iterator *end*, **Size** *steps*)

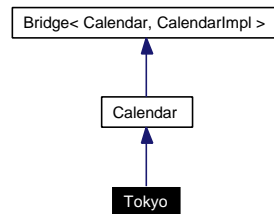
Time grid with mandatory time points.

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.

7.543 Tokyo Class Reference

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

Inheritance diagram for Tokyo:



7.543.1 Detailed Description

Tokyo calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Bank Holiday, January 2nd
- Bank Holiday, January 3rd
- Coming of Age Day, 2nd Monday in January
- National Foundation Day, February 11th
- Vernal Equinox
- Greenery Day, April 29th
- Constitution Memorial Day, May 3rd
- Holiday for a Nation, May 4th
- Children's Day, May 5th
- Marine Day, 3rd Monday in July
- Respect for the Aged Day, 3rd Monday in September
- Autumnal Equinox
- Health and Sports Day, 2nd Monday in October
- National Culture Day, November 3rd
- Labor Thanksgiving Day, November 23rd
- Emperor's Birthday, December 23rd
- Bank Holiday, December 31st

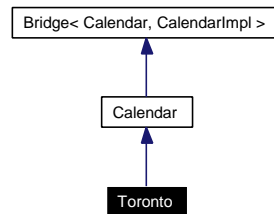
- a few one-shot holidays

Holidays falling on a Sunday are observed on the Monday following except for the bank holidays associated with the new year.

7.544 Toronto Class Reference

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

Inheritance diagram for Toronto:



7.544.1 Detailed Description

Toronto calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Victoria Day, The Monday on or preceding 24 May
- Canada Day, July 1st (possibly moved to Monday)
- Provincial Holiday, first Monday of August
- Labour Day, first Monday of September
- Thanksgiving Day, second Monday of October
- Remembrance Day, November 11th
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

7.545 TqrEigenDecomposition Class Reference

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

7.545.1 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** }

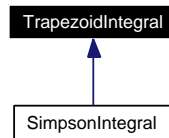
Public Member Functions

- **TqrEigenDecomposition** (const [Array](#) &diag, const [Array](#) &sub, EigenVectorCalculation calc=WithEigenVector, ShiftStrategy strategy=CloseEigenValue)
- const [Array](#) & **eigenvalues** () const
- const [Matrix](#) & **eigenvectors** () const
- [Size](#) **iterations** () const

7.546 TrapezoidIntegral Class Reference

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

Inheritance diagram for TrapezoidIntegral:



7.546.1 Detailed Description

Integral of a one-dimensional function.

Given a target accuracy ϵ , the integral of a function f between a and b is calculated by means of the trapezoid formula

$$\int_a^b f dx = \frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \dots + f(x_{N-1}) + \frac{1}{2}f(x_N)$$

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

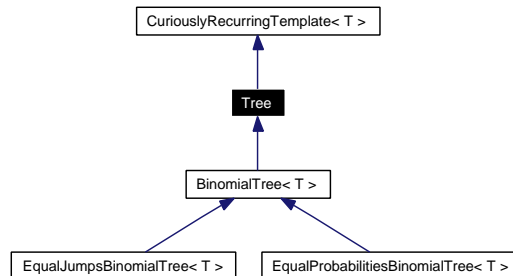
Protected Attributes

- [Real](#) accuracy_
- Method `method_`
- [Size](#) maxIterations_

7.547 Tree Class Template Reference

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

Inheritance diagram for Tree:



7.547.1 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...)

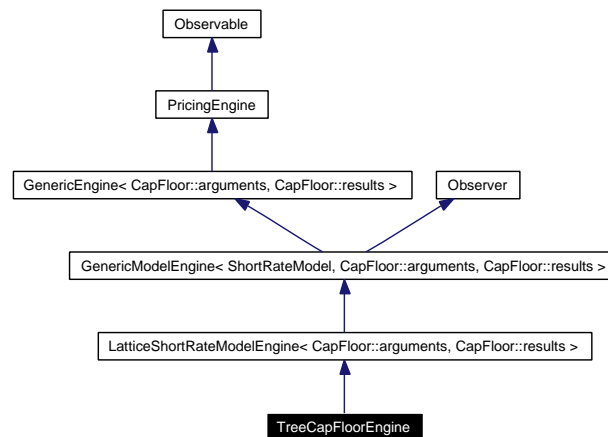
Public Member Functions

- **Tree** ([Size](#) columns)
- [Size](#) columns () const

7.548 TreeCapFloorEngine Class Reference

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

Inheritance diagram for TreeCapFloorEngine:



7.548.1 Detailed Description

Numerical lattice engine for cap/floors.

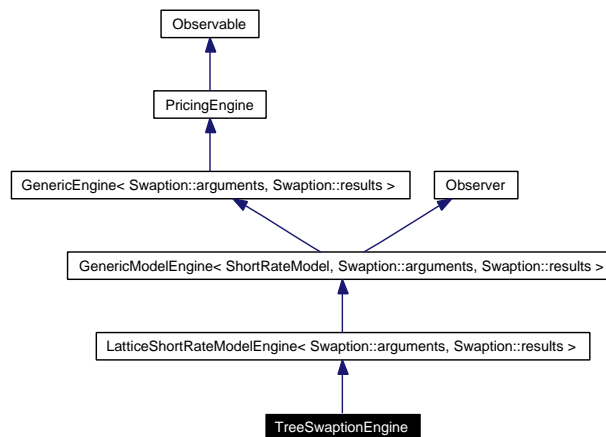
Public Member Functions

- **TreeCapFloorEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, [Size](#) timeSteps)
- **TreeCapFloorEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

7.549 TreeSwaptionEngine Class Reference

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

Inheritance diagram for TreeSwaptionEngine:



7.549.1 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. 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](#).

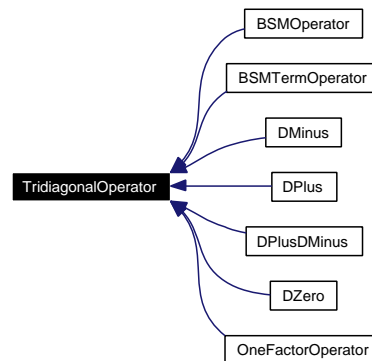
Public Member Functions

- **TreeSwaptionEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, [Size](#) timeSteps)
- **TreeSwaptionEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, const [Time-Grid](#) &timeGrid)
- void **calculate** () const

7.550 TridiagonalOperator Class Reference

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

Inheritance diagram for TridiagonalOperator:



7.550.1 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

7.551 TridiagonalOperator::TimeSetter Class Reference

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

7.551.1 Detailed Description

encapsulation of time-setting logic

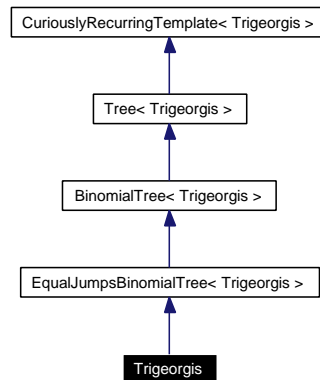
Public Member Functions

- virtual void **setTime** ([Time](#) t, [TridiagonalOperator](#) &L) const =0

7.552 Trigeorgis Class Reference

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

Inheritance diagram for Trigeorgis:



7.552.1 Detailed Description

Trigeorgis (additive equal jumps) binomial tree

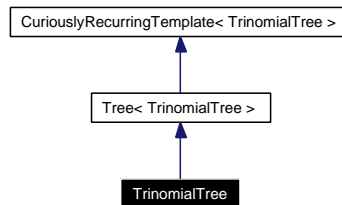
Public Member Functions

- **Trigeorgis** (const boost::shared_ptr< [StochasticProcess1D](#) > &process, [Time](#) end, [Size](#) steps, [Real](#) strike)

7.553 TrinomialTree Class Reference

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

Inheritance diagram for TrinomialTree:



7.553.1 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

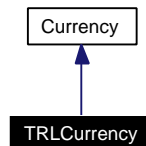
Protected Attributes

- std::vector< [Branching](#) > **branchings_**
- [Real](#) **x0_**
- std::vector< [Real](#) > **dx_**
- [TimeGrid](#) **timeGrid_**

7.554 TRLCurrency Class Reference

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

Inheritance diagram for TRLCurrency:



7.554.1 Detailed Description

Turkish lira.

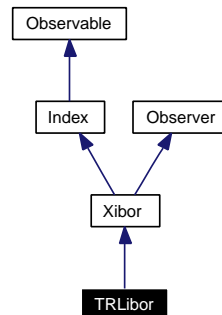
The ISO three-letter code is TRL; the numeric code is 792. It is divided in 100 kuruş.

Obsoleted by the new Turkish lira since 2005.

7.555 TRLibor Class Reference

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

Inheritance diagram for TRLibor:



7.555.1 Detailed Description

TRY LIBOR rate

TRY LIBOR fixed by TBA.

See <<http://www.trlibor.org/trlibor/english/default.asp>>

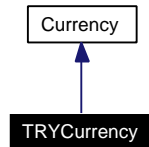
Public Member Functions

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

7.556 TRYCurrency Class Reference

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

Inheritance diagram for TRYCurrency:



7.556.1 Detailed Description

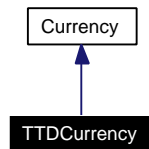
New Turkish lira.

The ISO three-letter code is TRY; the numeric code is 949. It is divided in 100 new kurus.

7.557 TTDCurrency Class Reference

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

Inheritance diagram for TTDCurrency:



7.557.1 Detailed Description

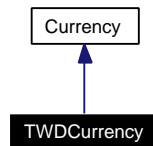
Trinidad & Tobago dollar.

The ISO three-letter code is TTD; the numeric code is 780. It is divided in 100 cents.

7.558 TWDCurrency Class Reference

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

Inheritance diagram for TWDCurrency:



7.558.1 Detailed Description

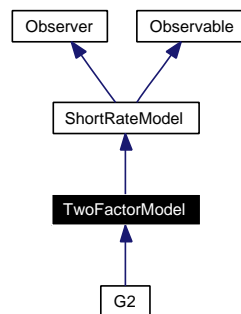
[Taiwan](#) dollar.

The ISO three-letter code is TWD; the numeric code is 901. It is divided in 100 cents.

7.559 TwoFactorModel Class Reference

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

Inheritance diagram for TwoFactorModel:



7.559.1 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.

7.560 TwoFactorModel::ShortRateDynamics Class Reference

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

7.560.1 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

$$dW_t^x dW_t^y = \rho dt$$

.

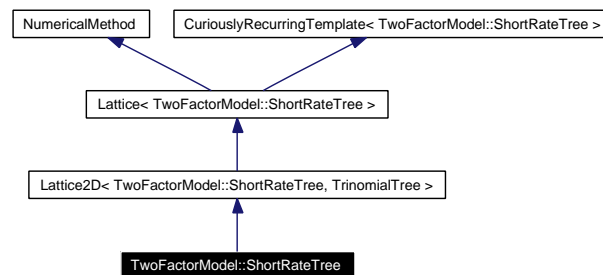
Public Member Functions

- **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 ρ between the two brownian motions.

7.561 TwoFactorModel::ShortRateTree Class Reference

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

Inheritance diagram for TwoFactorModel::ShortRateTree:



7.561.1 Detailed Description

Recombining two-dimensional tree discretizing the state variable.

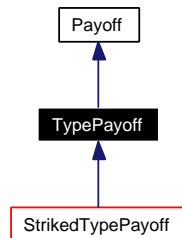
Public Member Functions

- [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

7.562 TypePayoff Class Reference

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

Inheritance diagram for TypePayoff:



7.562.1 Detailed Description

Intermediate class for call/put/straddle payoffs.

Public Member Functions

- **TypePayoff** (Option::Type type)
- Option::Type **optionType** () const

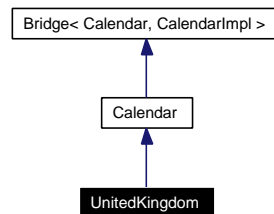
Protected Attributes

- Option::Type **type_**

7.563 UnitedKingdom Class Reference

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

Inheritance diagram for UnitedKingdom:



7.563.1 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)

7.563.2 Member Enumeration Documentation

7.563.2.1 enum [Market](#)

UK calendars.

Enumerator:

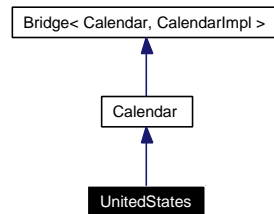
Settlement generic settlement calendar

Exchange London stock-exchange calendar.

7.564 UnitedStates Class Reference

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

Inheritance diagram for UnitedStates:



7.564.1 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`, `GovernmentBond` }
- US calendars.*

Public Member Functions

- `UnitedStates` (`Market` market=`Settlement`)

7.564.2 Member Enumeration Documentation

7.564.2.1 enum [Market](#)

US calendars.

Enumerator:

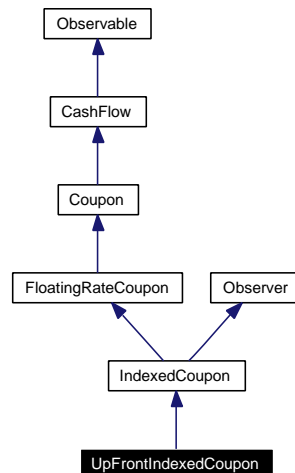
Settlement generic settlement calendar

Exchange New York stock-exchange calendar.

7.565 UpFrontIndexedCoupon Class Reference

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

Inheritance diagram for UpFrontIndexedCoupon:



7.565.1 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* &)

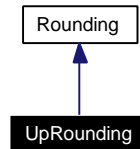
Protected Attributes

- *Calendar* **calendar_**

7.566 UpRounding Class Reference

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

Inheritance diagram for UpRounding:



7.566.1 Detailed Description

Up-rounding.

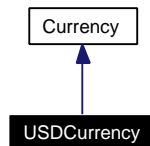
Public Member Functions

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

7.567 USDCurrency Class Reference

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

Inheritance diagram for USDCurrency:



7.567.1 Detailed Description

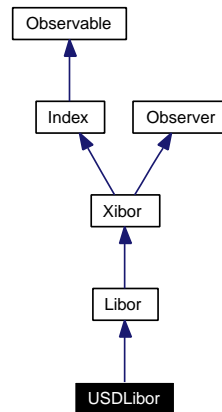
U.S. dollar.

The ISO three-letter code is USD; the numeric code is 840. It is divided in 100 cents.

7.568 USDLibor Class Reference

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

Inheritance diagram for USDLibor:



7.568.1 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>>.

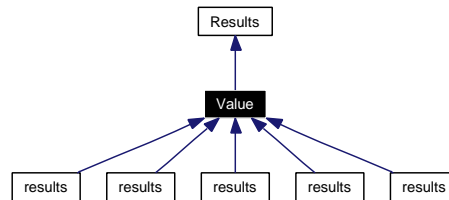
Public Member Functions

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

7.569 Value Class Reference

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

Inheritance diagram for Value:



7.569.1 Detailed Description

pricing results

Public Member Functions

- void `reset()`

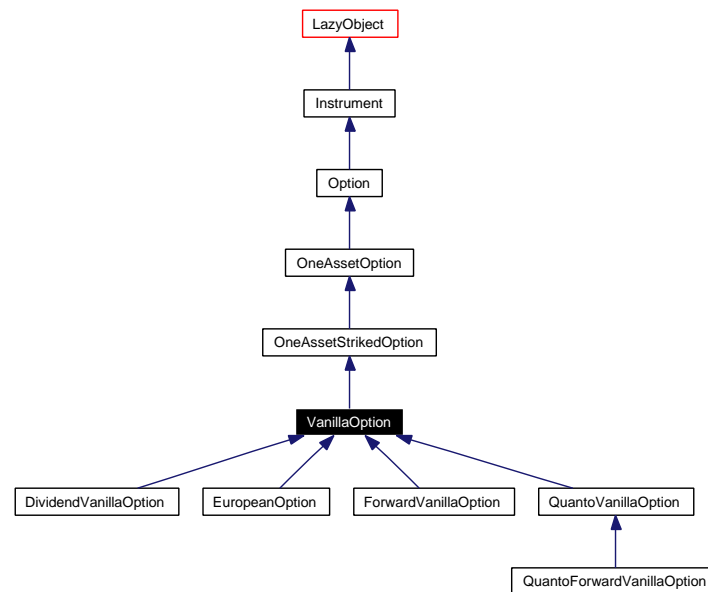
Public Attributes

- [Real](#) `value`
- [Real](#) `errorEstimate`

7.570 VanillaOption Class Reference

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

Inheritance diagram for VanillaOption:



7.570.1 Detailed Description

Vanilla option (no discrete dividends, no barriers) on a single asset.

Examples:

[AmericanOption.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](#) >())

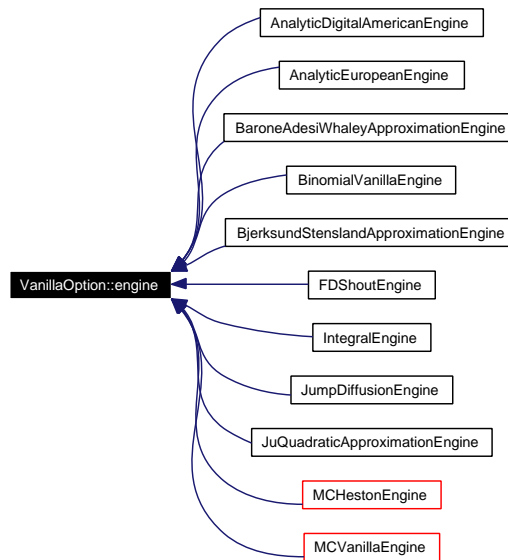
Classes

- class [engine](#)
Vanilla option engine base class.

7.571 VanillaOption::engine Class Reference

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

Inheritance diagram for VanillaOption::engine:



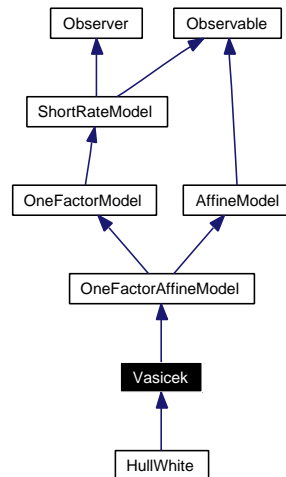
7.571.1 Detailed Description

Vanilla option engine base class.

7.572 Vasicek Class Reference

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

Inheritance diagram for Vasicek:



7.572.1 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 σ are constants; a risk premium λ 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.

7.573 Vasicek::Dynamics Class Reference

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

7.573.1 Detailed Description

Short-rate dynamics in the Vasicek model.

The short-rate follows an Ornstein-Uhlenbeck process with mean b .

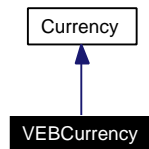
Public Member Functions

- **Dynamics** ([Real](#) a, [Real](#) b, [Real](#) sigma, [Real](#) r0)
- virtual [Real](#) **variable** ([Time](#), [Rate](#) r) const
- virtual [Real](#) **shortRate** ([Time](#), [Real](#) x) const

7.574 VEBCurrency Class Reference

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

Inheritance diagram for VEBCurrency:



7.574.1 Detailed Description

Venezuelan bolivar.

The ISO three-letter code is VEB; the numeric code is 862. It is divided in 100 centimos.

7.575 Visitor Class Template Reference

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

7.575.1 Detailed Description

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

Visitor for a specific class

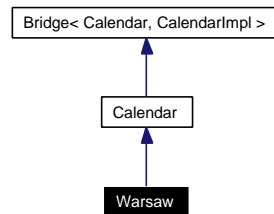
Public Member Functions

- virtual void **visit** (T &)=0

7.576 Warsaw Class Reference

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

Inheritance diagram for Warsaw:



7.576.1 Detailed Description

Warsaw calendar

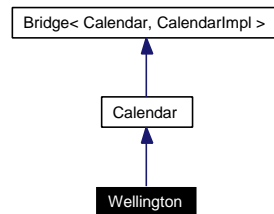
Holidays:

- Saturdays
- Sundays
- Easter Monday
- Corpus Christi
- New Year's Day, January 1st
- May Day, May 1st
- Constitution Day, May 3rd
- Assumption of the Blessed Virgin Mary, August 15th
- All Saints Day, November 1st
- Independence Day, November 11th
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

7.577 Wellington Class Reference

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

Inheritance diagram for Wellington:



7.577.1 Detailed Description

Wellington calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday or Tuesday)
- Day after New Year's Day, January 2st (possibly moved to Monday or Tuesday)
- Anniversary Day, Monday nearest January 22nd
- Waitangi Day. February 6th
- Good Friday
- Easter Monday
- ANZAC Day. April 25th
- Queen's Birthday, first Monday in June
- Labour Day, fourth Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

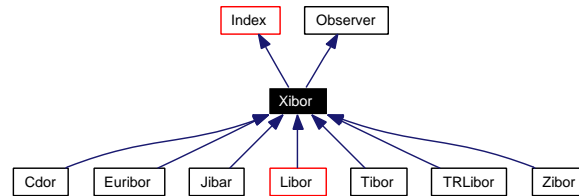
Note:

The holiday rules for [Wellington](http://www.jrefinery.com/ibd/) were documented by David Gilbert for IDB (<http://www.jrefinery.com/ibd/>)

7.578 Xibor Class Reference

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

Inheritance diagram for Xibor:



7.578.1 Detailed Description

base class for LIBOR-like indexes

Public Member Functions

- **Xibor** (const std::string &familyName, Integer n, TimeUnit units, Integer settlementDays, const Currency ¤cy, 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_`

7.578.2 Member Function Documentation

7.578.2.1 `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](#).

7.578.2.2 `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](#).

7.578.2.3 `std::string name () const` [virtual]

Returns the name of the index.

Warning:

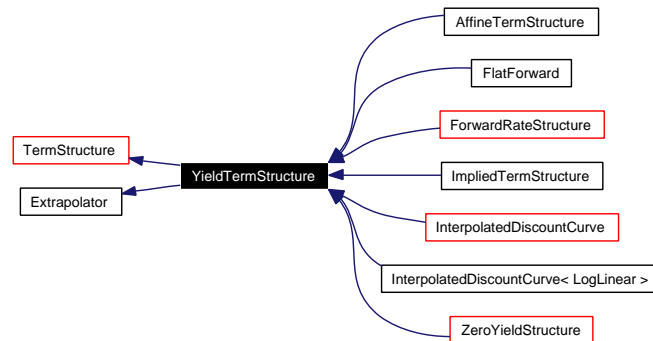
This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implements [Index](#).

7.579 YieldTermStructure Class Reference

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

Inheritance diagram for YieldTermStructure:



7.579.1 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** > ×, **Frequency** freq=Annual, bool extrapolate=false) const
- **Rate parRate** (**Year** tenor, **Time** t0, **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

7.579.2 Constructor & Destructor Documentation

7.579.2.1 [YieldTermStructure](#) ()

default constructor

Warning:

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

7.579.3 Member Function Documentation

7.579.3.1 [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.

7.579.3.2 [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.

7.579.3.3 [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.

7.579.3.4 [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.

7.579.3.5 [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.

7.579.3.6 **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.

7.579.3.7 **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.

7.579.3.8 **Rate** parRate (**Year** *tenor*, **Time** *t0*, **Frequency** *freq* = Annual, bool *extrapolate* = false) const

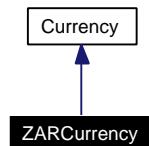
Deprecated

use the overload taking a vector of times

7.580 ZARCurrency Class Reference

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

Inheritance diagram for ZARCurrency:



7.580.1 Detailed Description

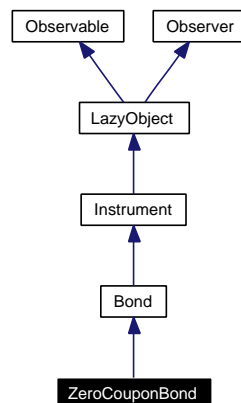
South-African rand.

The ISO three-letter code is ZAR; the numeric code is 710. It is divided into 100 cents.

7.581 ZeroCouponBond Class Reference

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

Inheritance diagram for ZeroCouponBond:



7.581.1 Detailed Description

zero-coupon bond

Tests

calculations are tested by checking results against cached values.

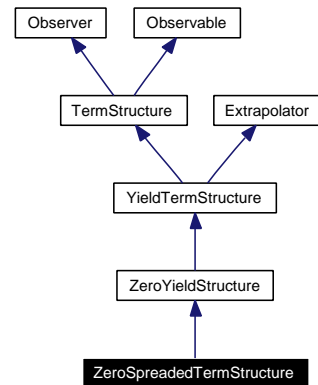
Public Member Functions

- **ZeroCouponBond** (const [Date](#) &issueDate, const [Date](#) &maturityDate, [Integer](#) settlement-Days, const [DayCounter](#) &dayCounter, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention=Following, [Real](#) redemption=100.0, const [Handle](#)< [YieldTermStructure](#) > &discountCurve=[Handle](#)< [YieldTermStructure](#) >())

7.582 ZeroSpreadedTermStructure Class Reference

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

Inheritance diagram for ZeroSpreadedTermStructure:



7.582.1 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

7.582.2 Member Function Documentation

7.582.2.1 [Rate forwardImpl](#) (Time) const [protected]

returns the spreaded forward rate

Warning:

This method must disappear should the spread become a curve

7.583 ZeroYield Struct Reference

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

7.583.1 Detailed Description

Zero-curve traits.

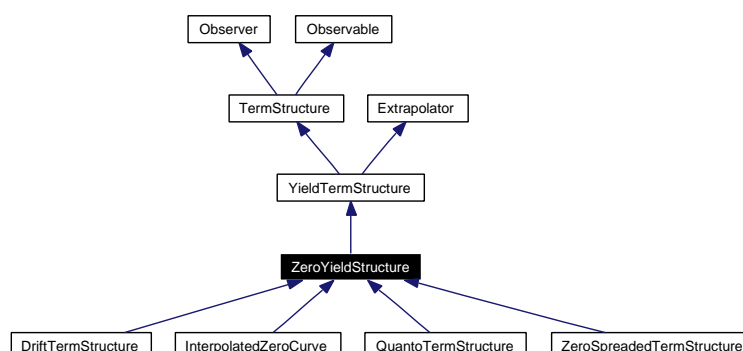
Static Public Member Functions

- 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)

7.584 ZeroYieldStructure Class Reference

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

Inheritance diagram for ZeroYieldStructure:



7.584.1 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*

7.584.2 Member Function Documentation

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

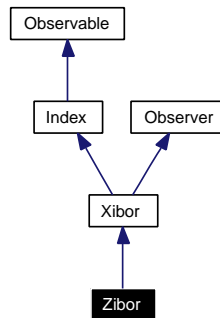
Returns the discount factor for the given date calculating it from the zero yield.

Implements [YieldTermStructure](#).

7.585 Zibor Class Reference

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

Inheritance diagram for Zibor:



7.585.1 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.

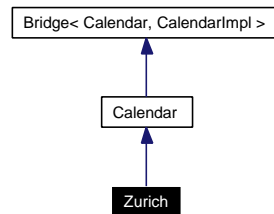
Public Member Functions

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

7.586 Zurich Class Reference

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

Inheritance diagram for Zurich:



7.586.1 Detailed Description

Zurich calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Berchtoldstag, January 2nd
- Good Friday
- Easter Monday
- Ascension Day
- Whit Monday
- Labour Day, May 1st
- National Day, August 1st
- Christmas, December 25th
- St. Stephen's Day, December 26th

Chapter 8

QuantLib File Documentation

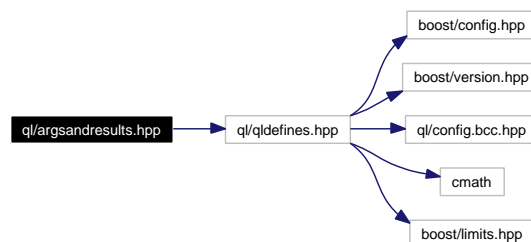
8.1 ql/argsandresults.hpp File Reference

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

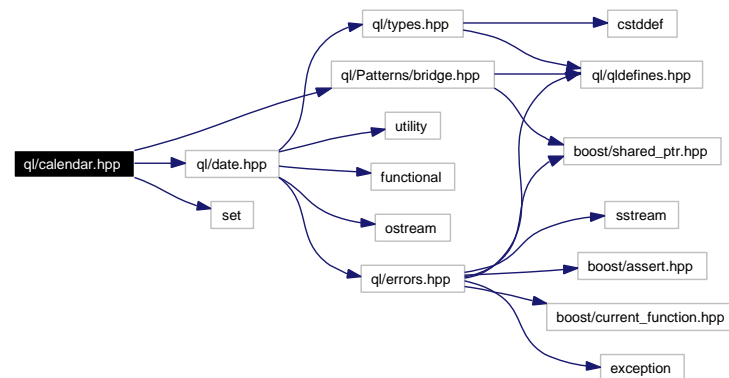
calendar class

```
#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

Enumerations

- enum [QuantLib::BusinessDayConvention](#) {
[QuantLib::Unadjusted](#), [QuantLib::Preceding](#), [QuantLib::ModifiedPreceding](#), [QuantLib::Following](#),
[QuantLib::ModifiedFollowing](#), [QuantLib::MonthEndReference](#) }
Business Day conventions.

8.3 ql/Calendars/beijing.hpp File Reference

8.3.1 Detailed Description

Beijing calendar.

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

Include dependency graph for beijing.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Beijing](#)
Beijing calendar

8.4 ql/Calendars/bombay.hpp File Reference

8.4.1 Detailed Description

Bombay calendar.

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

Include dependency graph for `bombay.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class **Bombay**
Bombay calendar

8.5 ql/Calendars/bratislava.hpp File Reference

8.5.1 Detailed Description

Bratislava calendar.

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

Include dependency graph for bratislava.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Bratislava](#)
Bratislava calendar

8.6 ql/Calendars/budapest.hpp File Reference

8.6.1 Detailed Description

Budapest calendar.

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

Include dependency graph for budapest.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Budapest**
Budapest calendar

8.7 ql/Calendars/copenhagen.hpp File Reference

8.7.1 Detailed Description

Copenhagen calendar.

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

Include dependency graph for copenhagen.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Copenhagen](#)
Copenhagen calendar

8.8 ql/Calendars/germany.hpp File Reference

8.8.1 Detailed Description

German calendars.

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

Include dependency graph for germany.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Germany**
German calendars.

8.9 ql/Calendars/helsinki.hpp File Reference

8.9.1 Detailed Description

Helsinki calendar.

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

Include dependency graph for helsinki.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Helsinki](#)
Helsinki calendar

8.10 ql/Calendars/hongkong.hpp File Reference

8.10.1 Detailed Description

Hong Kong calendar.

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

Include dependency graph for hongkong.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [HongKong](#)
Hong Kong calendar.

8.11 ql/Calendars/istanbul.hpp File Reference

8.11.1 Detailed Description

Istanbul calendar.

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

Include dependency graph for istanbul.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Istanbul](#)
Istanbul calendar

8.12 ql/Calendars/italy.hpp File Reference

8.12.1 Detailed Description

Italian calendars.

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

Include dependency graph for italy.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Italy](#)
Italian calendars.

8.13 ql/Calendars/johannesburg.hpp File Reference

8.13.1 Detailed Description

Johannesburg calendar.

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

Include dependency graph for johannesburg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Johannesburg](#)
Johannesburg calendar

8.14 ql/Calendars/jointcalendar.hpp File Reference

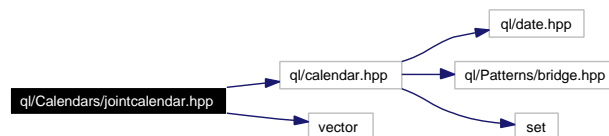
8.14.1 Detailed Description

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

8.15 ql/Calendars/nullcalendar.hpp File Reference

8.15.1 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.

8.16 ql/Calendars/oslo.hpp File Reference

8.16.1 Detailed Description

Oslo calendar.

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

Include dependency graph for oslo.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Oslo](#)
Oslo calendar

8.17 ql/Calendars/prague.hpp File Reference

8.17.1 Detailed Description

Prague calendar.

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

Include dependency graph for prague.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Prague](#)
Prague calendar

8.18 ql/Calendars/riyadh.hpp File Reference

8.18.1 Detailed Description

Riyadh calendar.

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

Include dependency graph for riyadh.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Riyadh](#)
Riyadh calendar

8.19 ql/Calendars/seoul.hpp File Reference

8.19.1 Detailed Description

South Korea calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for seoul.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Seoul](#)
Seoul calendar

8.20 ql/Calendars/singapore.hpp File Reference

8.20.1 Detailed Description

Singapore calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for singapore.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Singapore](#)
Singapore calendar

8.21 ql/Calendars/stockholm.hpp File Reference

8.21.1 Detailed Description

Stockholm calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for stockholm.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Stockholm](#)
Stockholm calendar

8.22 ql/Calendars/sydney.hpp File Reference

8.22.1 Detailed Description

Sydney calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for sydney.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Sydney](#)
Sydney calendar (New South Wales, Australia)

8.23 ql/Calendars/taipei.hpp File Reference

8.23.1 Detailed Description

Taipei calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for taipei.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Taipei](#)
Taipei calendar

8.24 ql/Calendars/taiwan.hpp File Reference

8.24.1 Detailed Description

Taiwan calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for taiwan.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Taiwan](#)
Taiwan calendar

8.25 ql/Calendars/target.hpp File Reference

8.25.1 Detailed Description

TARGET calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for target.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TARGET**
TARGET calendar

8.26 ql/Calendars/tokyo.hpp File Reference

8.26.1 Detailed Description

Tokyo calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for tokyo.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Tokyo**
Tokyo calendar

8.27 ql/Calendars/toronto.hpp File Reference

8.27.1 Detailed Description

Toronto calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for toronto.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Toronto](#)
Toronto calendar

8.28 ql/Calendars/unitedkingdom.hpp File Reference

8.28.1 Detailed Description

UK calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for unitedkingdom.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [UnitedKingdom](#)
United Kingdom calendars.

8.29 ql/Calendars/unitedstates.hpp File Reference

8.29.1 Detailed Description

US calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for unitedstates.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [UnitedStates](#)
United States calendars.

8.30 ql/Calendars/warsaw.hpp File Reference

8.30.1 Detailed Description

Warsaw calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for warsaw.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Warsaw](#)
Warsaw calendar

8.31 ql/Calendars/wellington.hpp File Reference

8.31.1 Detailed Description

Wellington calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for wellington.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Wellington](#)
Wellington calendar

8.32 ql/Calendars/zurich.hpp File Reference

8.32.1 Detailed Description

Zurich calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for zurich.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Zurich](#)
Zurich calendar

8.33 ql/capvolstructures.hpp File Reference

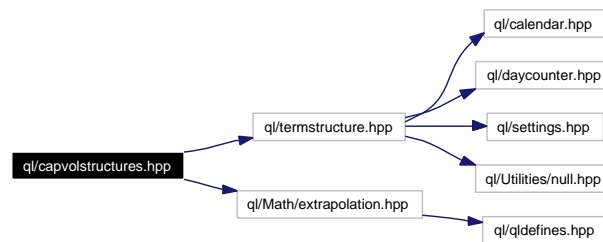
8.33.1 Detailed Description

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.

8.34 ql/cashflow.hpp File Reference

8.34.1 Detailed Description

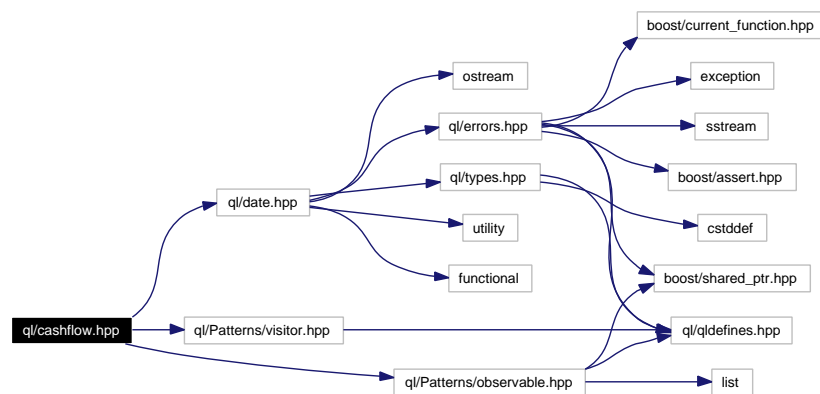
Base class for cash flows.

```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

```
#include <ql/Patterns/visitor.hpp>
```

Include dependency graph for cashflow.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CashFlow](#)
Base class for cash flows.

8.35 ql/CashFlows/analysis.hpp File Reference

8.35.1 Detailed Description

Cash-flow analysis functions.

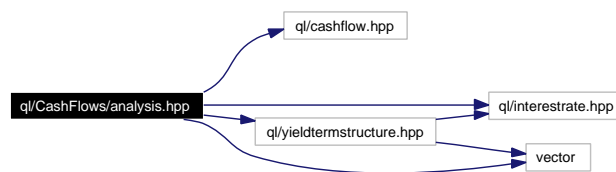
```
#include <ql/cashflow.hpp>
```

```
#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

8.36 ql/CashFlows/basispointsensitivity.hpp File Reference

8.36.1 Detailed Description

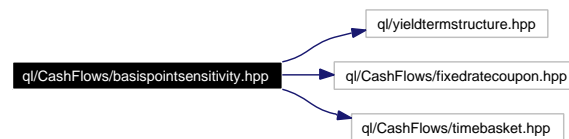
basis point sensitivity calculator

```
#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 [BPSCalculator](#)
basis point sensitivity (BPS) calculator
- 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)

8.37 ql/CashFlows/cashflowvectors.hpp File Reference

8.37.1 Detailed Description

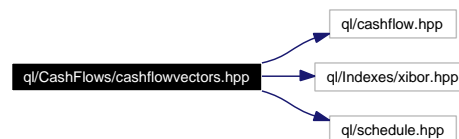
Cash flow vector builders.

```
#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

8.38 ql/CashFlows/coupon.hpp File Reference

8.38.1 Detailed Description

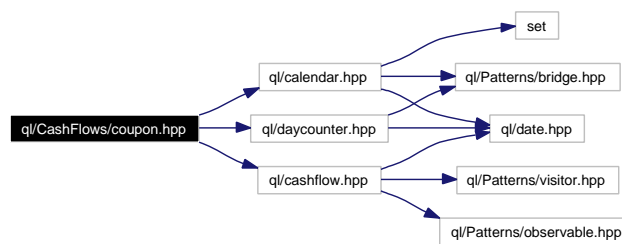
Coupon accruing over a fixed period.

```
#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

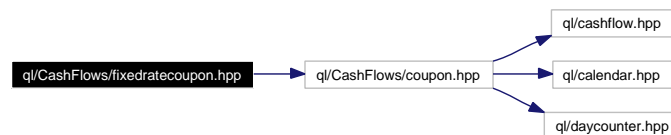
8.39 ql/CashFlows/fixedratecoupon.hpp File Reference

8.39.1 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

8.40 ql/CashFlows/floatingratecoupon.hpp File Reference

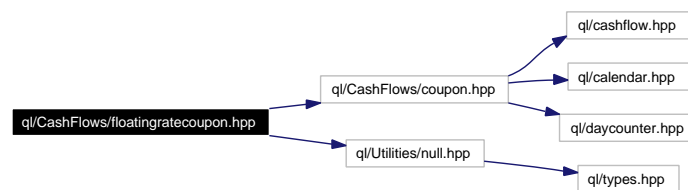
8.40.1 Detailed Description

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

8.41 ql/CashFlows/inarrearindexedcoupon.hpp File Reference

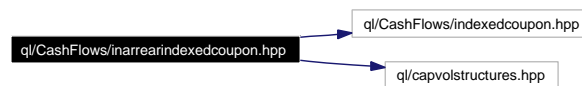
8.41.1 Detailed Description

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.

8.42 ql/CashFlows/indexedcashflowvectors.hpp File Reference

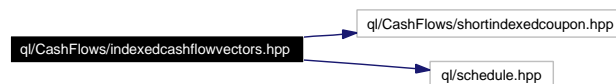
8.42.1 Detailed Description

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

8.43 ql/CashFlows/indexedcoupon.hpp File Reference

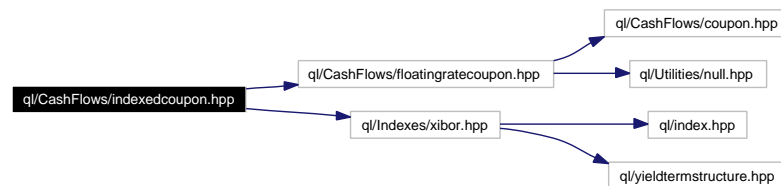
8.43.1 Detailed Description

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.

8.44 ql/CashFlows/parcoupon.hpp File Reference

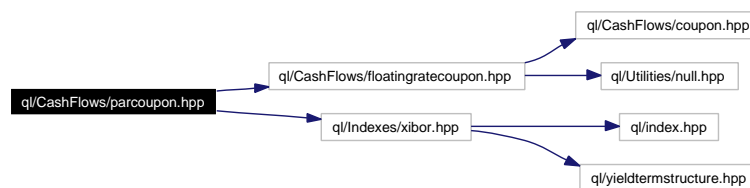
8.44.1 Detailed Description

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

8.45 ql/CashFlows/shortfloatingcoupon.hpp File Reference

8.45.1 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

8.46 ql/CashFlows/shortindexedcoupon.hpp File Reference

8.46.1 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

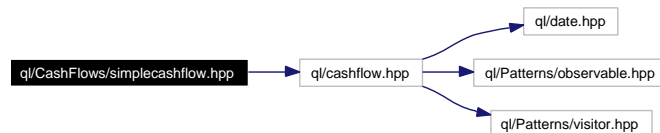
8.47 ql/CashFlows/simplecashflow.hpp File Reference

8.47.1 Detailed Description

Predetermined cash flow.

```
#include <ql/cashflow.hpp>
```

Include dependency graph for simplecashflow.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SimpleCashFlow](#)
Predetermined cash flow.

8.48 ql/CashFlows/timebasket.hpp File Reference

8.48.1 Detailed Description

Distribution over a number of dates

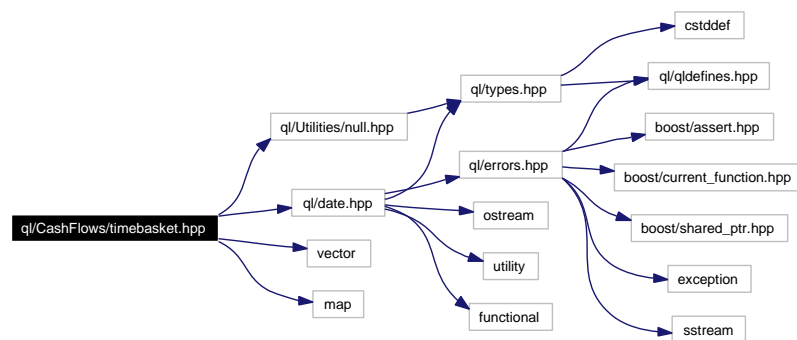
```
#include <ql/date.hpp>
```

```
#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.

8.49 ql/CashFlows/upfrontindexedcoupon.hpp File Reference

8.49.1 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

8.50 ql/Currencies/africa.hpp File Reference

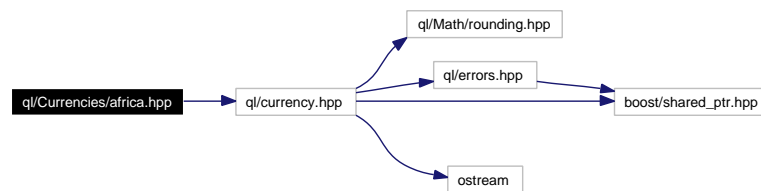
8.50.1 Detailed Description

African currencies.

Data from 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.

8.51 ql/Currencies/america.hpp File Reference

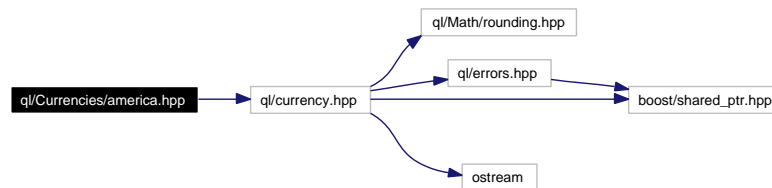
8.51.1 Detailed Description

American currencies.

Data from 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.

8.52 ql/Currencies/asia.hpp File Reference

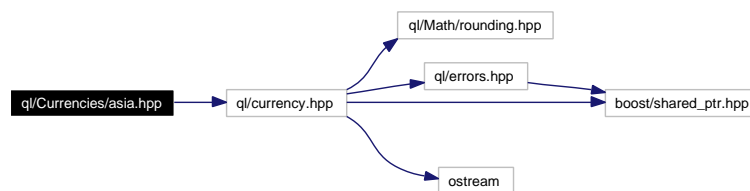
8.52.1 Detailed Description

Asian currencies.

Data from 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.

8.53 ql/Currencies/europe.hpp File Reference

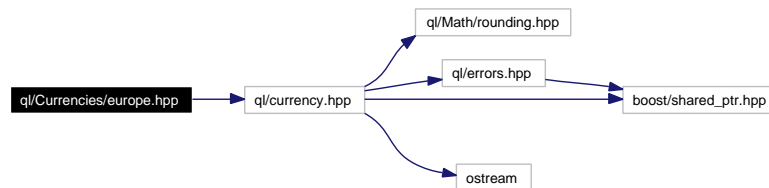
8.53.1 Detailed Description

European currencies.

Data from 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.

8.54 ql/Currencies/exchangeratemanager.hpp File Reference

8.54.1 Detailed Description

exchange-rate repository

```
#include <ql/exchangerate.hpp>
```

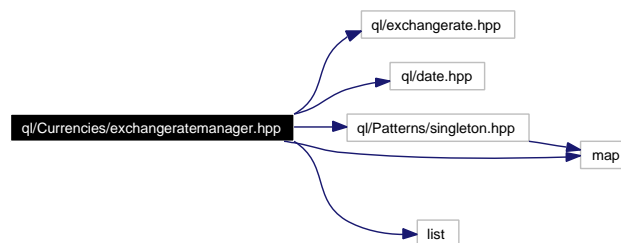
```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <list>
```

```
#include <map>
```

Include dependency graph for `exchangeratemanager.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [ExchangeRateManager](#)
exchange-rate repository

8.55 ql/Currencies/oceania.hpp File Reference

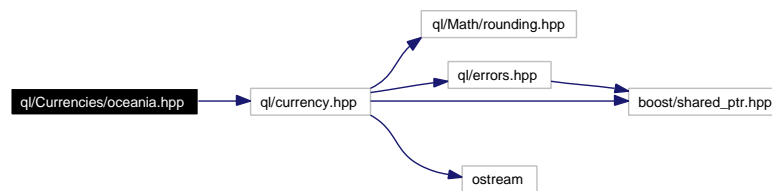
8.55.1 Detailed Description

Oceanian currencies.

Data from 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.

8.56 ql/currency.hpp File Reference

8.56.1 Detailed Description

Known currencies.

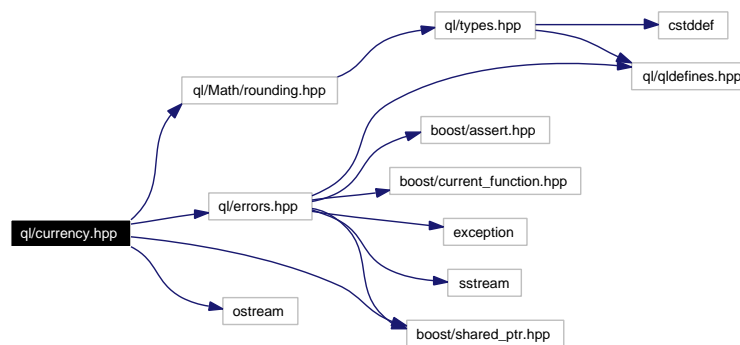
```
#include <ql/Math/rounding.hpp>
```

```
#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

8.57 ql/date.hpp File Reference

8.57.1 Detailed Description

date- and time-related classes, typedefs and enumerations

```
#include <ql/errors.hpp>
```

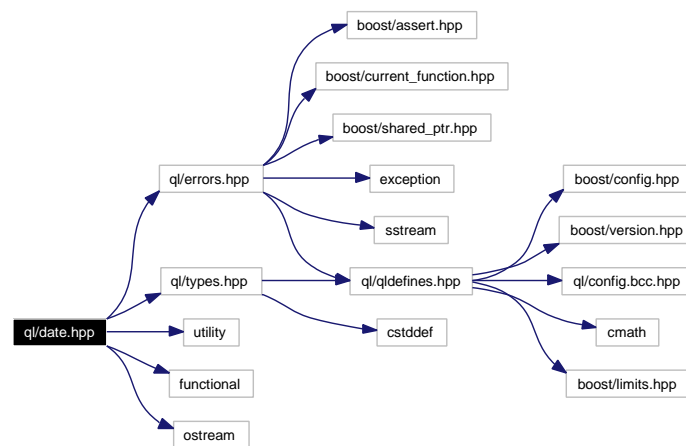
```
#include <ql/types.hpp>
```

```
#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::IMMMonth](#) { **H** = 3, **M** = 6, **U** = 9, **Z** = 12 }
Main cycle of the International Money Market (a.k.a. IMM) Months.
- 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)

8.58 ql/daycounter.hpp File Reference

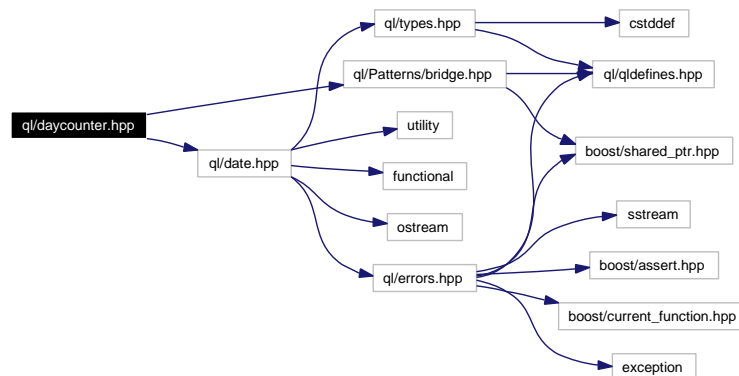
8.58.1 Detailed Description

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

8.59 ql/DayCounters/actual360.hpp File Reference

8.59.1 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.

8.60 ql/DayCounters/actual365fixed.hpp File Reference

8.60.1 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.

8.61 ql/DayCounters/actualactual.hpp File Reference

8.61.1 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.

8.62 ql/DayCounters/one.hpp File Reference

8.62.1 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

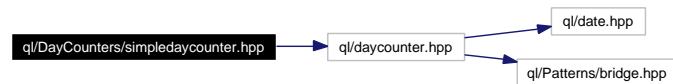
8.63 ql/DayCounters/simpliedaycounter.hpp File Reference

8.63.1 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.

8.64 ql/DayCounters/thirty360.hpp File Reference

8.64.1 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

8.65 ql/discretizedasset.hpp File Reference

8.65.1 Detailed Description

Discretized asset classes.

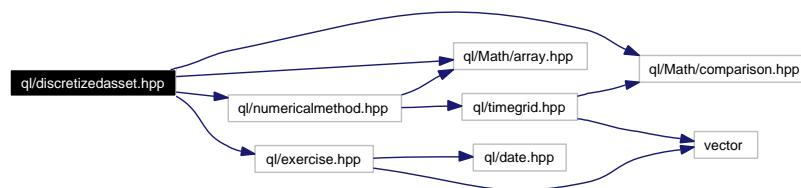
```
#include <ql/numericalmethod.hpp>
```

```
#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.

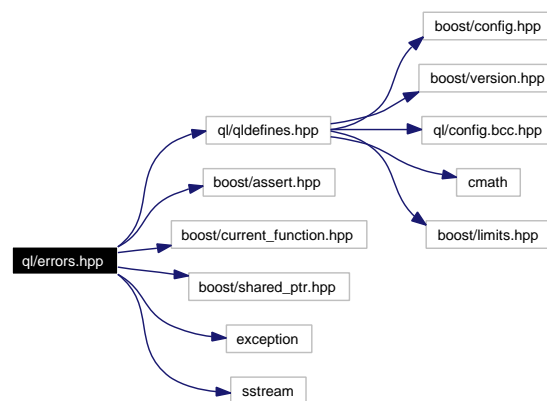
8.66 ql/errors.hpp File Reference

8.66.1 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

8.66.2 Define Documentation

8.66.2.1 `#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)

8.66.2.2 `#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

8.66.2.3 `#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](#).

8.66.2.4 #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

8.67 ql/exchangerate.hpp File Reference

8.67.1 Detailed Description

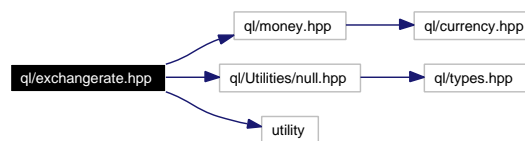
exchange rate between two currencies

```
#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

8.68 ql/exercise.hpp File Reference

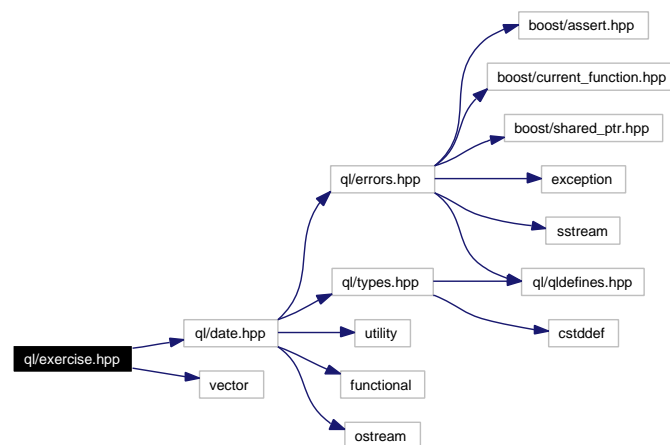
8.68.1 Detailed Description

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.

8.69 ql/FiniteDifferences/americancondition.hpp File Reference

8.69.1 Detailed Description

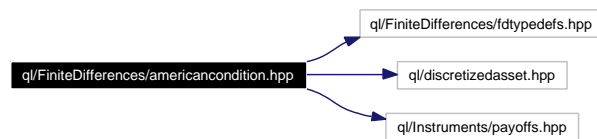
american option exercise condition

```
#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.

8.70 ql/FiniteDifferences/boundarycondition.hpp File Reference

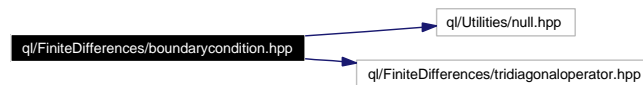
8.70.1 Detailed Description

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).

8.71 ql/FiniteDifferences/bsmoperator.hpp File Reference

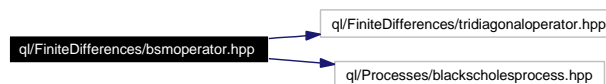
8.71.1 Detailed Description

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.

8.72 ql/FiniteDifferences/bsmtermoperator.hpp File Reference

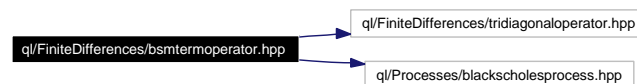
8.72.1 Detailed Description

differential operator for Black-Scholes-Merton equation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for bsmtermoperator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BSMTermOperator](#)
Black-Scholes-Merton differential operator.

8.73 ql/FiniteDifferences/cranknicolson.hpp File Reference

8.73.1 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.

8.74 ql/FiniteDifferences/dminus.hpp File Reference

8.74.1 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

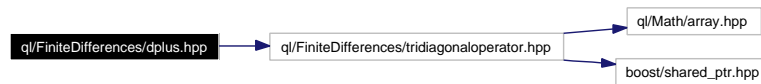
8.75 ql/FiniteDifferences/dplus.hpp File Reference

8.75.1 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

8.76 ql/FiniteDifferences/dplusdminus.hpp File Reference

8.76.1 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

8.77 ql/FiniteDifferences/dzero.hpp File Reference

8.77.1 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

8.78 ql/FiniteDifferences/expliciteuler.hpp File Reference

8.78.1 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.

8.79 ql/FiniteDifferences/fdtypedefs.hpp File Reference

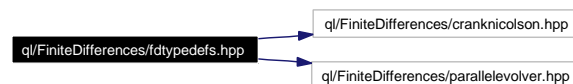
8.79.1 Detailed Description

default choices for template instantiations

```
#include <ql/FiniteDifferences/cranknicolson.hpp>
```

```
#include <ql/FiniteDifferences/parallelevolver.hpp>
```

Include dependency graph for fdtypedefs.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef `FiniteDifferenceModel< CrankNicolson< TridiagonalOperator > >` [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

8.80 ql/FiniteDifferences/finitedifferencemodel.hpp File Reference

8.80.1 Detailed Description

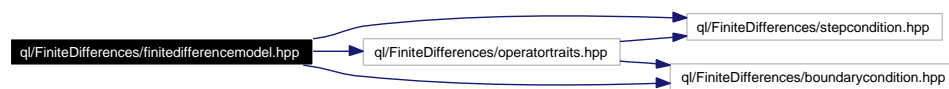
generic finite difference model

```
#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.

8.81 ql/FiniteDifferences/impliciteuler.hpp File Reference

8.81.1 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.

8.82 ql/FiniteDifferences/mixedscheme.hpp File Reference

8.82.1 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.

8.83 ql/FiniteDifferences/onefactoroperator.hpp File Reference

8.83.1 Detailed Description

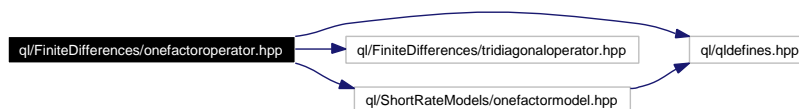
general differential operator for one-factor interest rate models

```
#include <ql/qldefines.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for onefactoroperator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OneFactorOperator](#)
Interest-rate single factor model differential operator.

8.84 ql/FiniteDifferences/operatortraits.hpp File Reference

8.84.1 Detailed Description

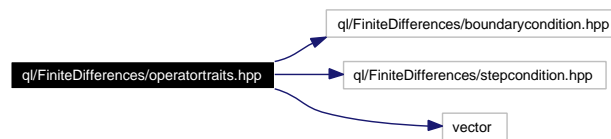
Differential operator traits.

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

```
#include <vector>
```

Include dependency graph for operatortraits.hpp:



Namespaces

- namespace **QuantLib**

8.85 ql/FiniteDifferences/parallelevolver.hpp File Reference

8.85.1 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.

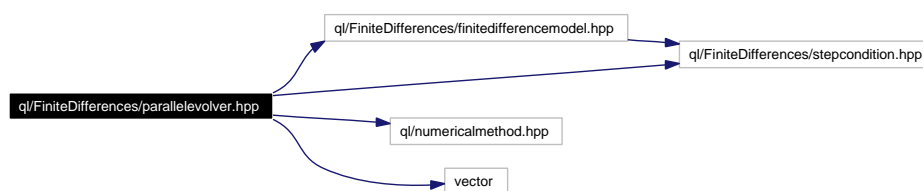
```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
```

```
#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.

8.86 ql/FiniteDifferences/shoutcondition.hpp File Reference

8.86.1 Detailed Description

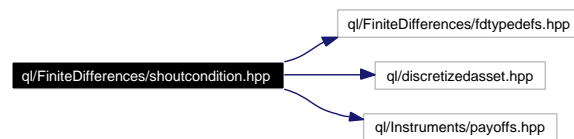
shout option exercise condition

```
#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.

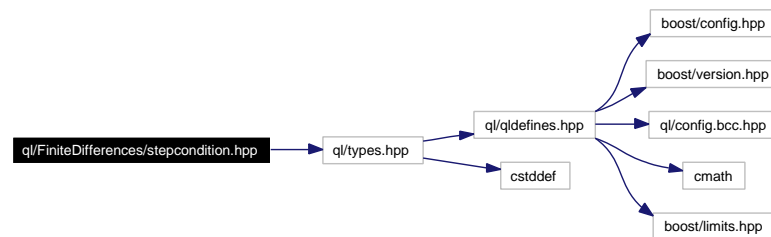
8.87 ql/FiniteDifferences/stepcondition.hpp File Reference

8.87.1 Detailed Description

conditions to be applied at every time step

```
#include <ql/types.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

8.88 ql/FiniteDifferences/tridiagonaloperator.hpp File Reference

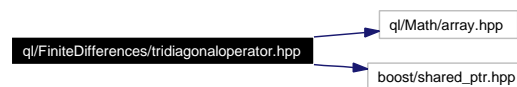
8.88.1 Detailed Description

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)

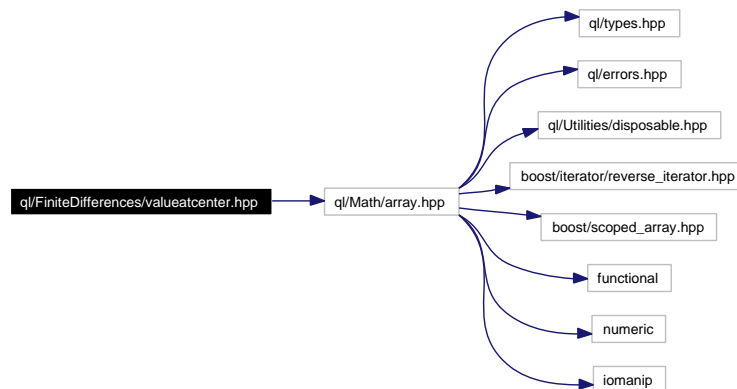
8.89 ql/FiniteDifferences/valueatcenter.hpp File Reference

8.89.1 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

- [Real QuantLib::valueAtCenter](#) (const Array &a)
- [Real QuantLib::firstDerivativeAtCenter](#) (const Array &a, const Array &grid)
- [Real QuantLib::secondDerivativeAtCenter](#) (const Array &a, const Array &grid)

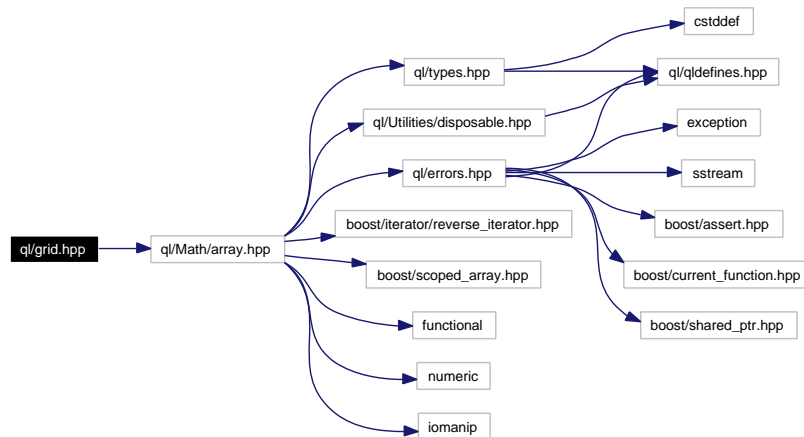
8.90 ql/grid.hpp File Reference

8.90.1 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)

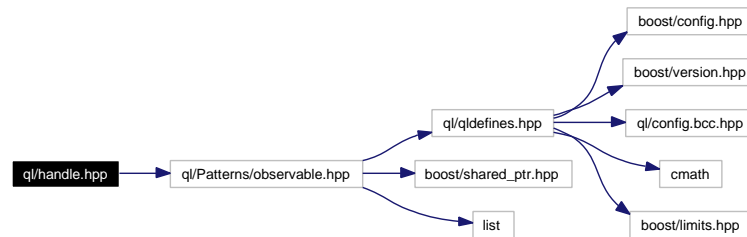
8.91 ql/handle.hpp File Reference

8.91.1 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.

8.92 ql/history.hpp File Reference

8.92.1 Detailed Description

history class

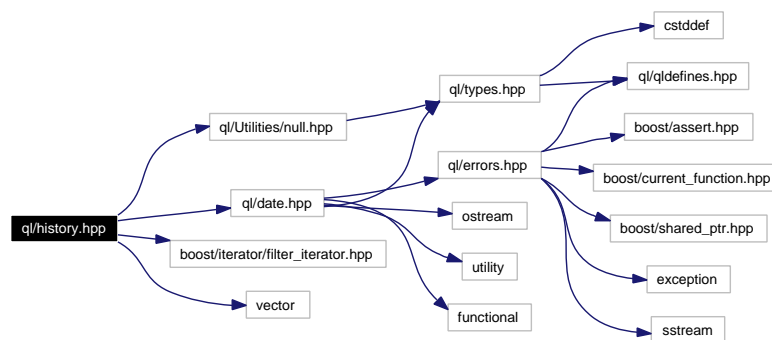
```
#include <ql/date.hpp>
```

```
#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

8.93 ql/index.hpp File Reference

8.93.1 Detailed Description

purely virtual base class for indexes

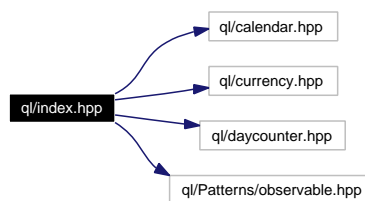
```
#include <ql/calendar.hpp>
```

```
#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

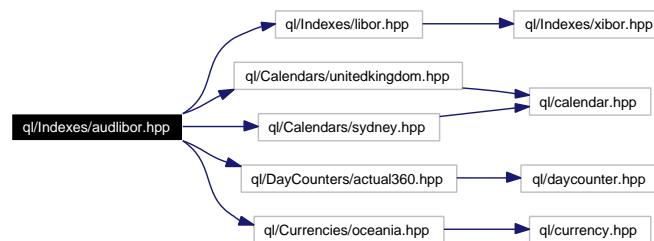
8.94 ql/Indexes/audlibor.hpp File Reference

8.94.1 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

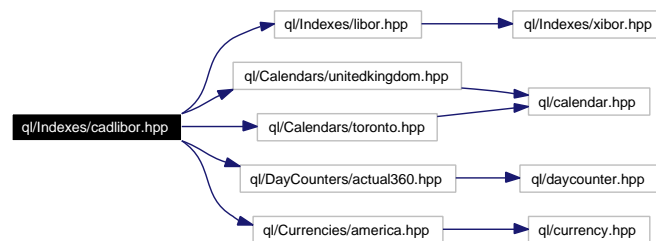
8.95 ql/Indexes/cadlibor.hpp File Reference

8.95.1 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

8.96 ql/Indexes/cdor.hpp File Reference

8.96.1 Detailed Description

CDOR rate

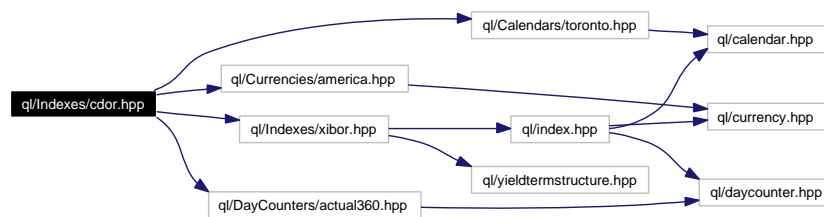
```
#include <ql/Indexes/xibor.hpp>
```

```
#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

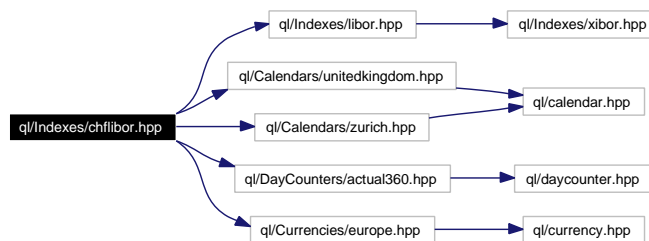
8.97 ql/Indexes/chflibor.hpp File Reference

8.97.1 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

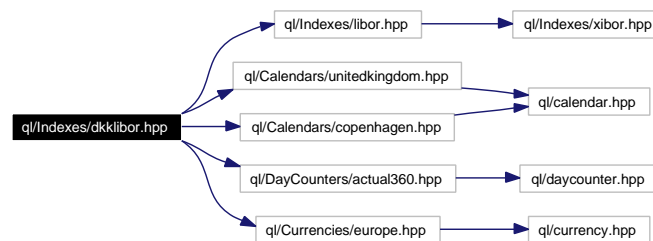
8.98 ql/Indexes/dkklibor.hpp File Reference

8.98.1 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

8.99 ql/Indexes/euribor.hpp File Reference

8.99.1 Detailed Description

Euribor index

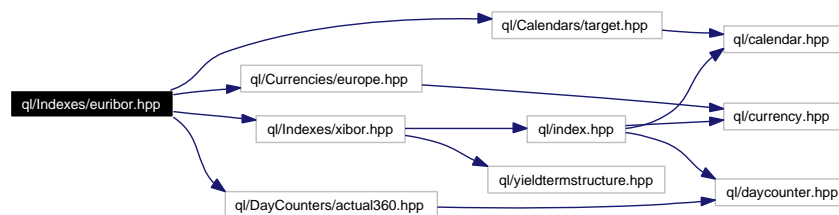
```
#include <ql/Indexes/xibor.hpp>
```

```
#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

8.100 ql/Indexes/eurlibor.hpp File Reference

8.100.1 Detailed Description

EUR LIBOR rate

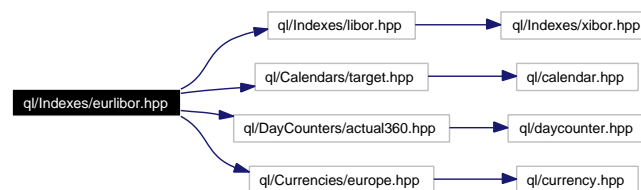
```
#include <ql/Indexes/libor.hpp>
```

```
#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

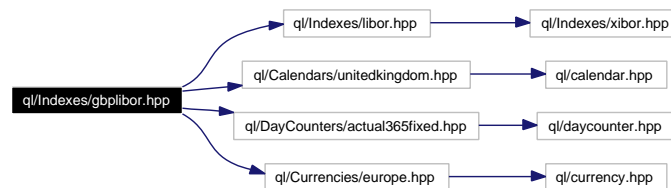
8.101 ql/Indexes/gbplibor.hpp File Reference

8.101.1 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

8.102 ql/Indexes/indexmanager.hpp File Reference

8.102.1 Detailed Description

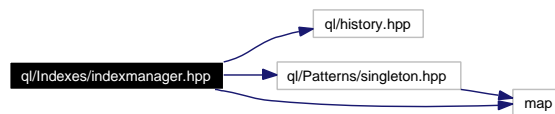
global repository for past index fixings

```
#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

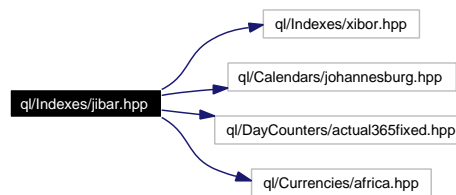
8.103 ql/Indexes/jibar.hpp File Reference

8.103.1 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

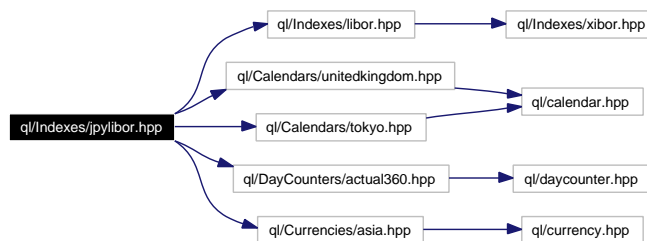
8.104 ql/Indexes/jpylibor.hpp File Reference

8.104.1 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

8.105 ql/Indexes/libor.hpp File Reference

8.105.1 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

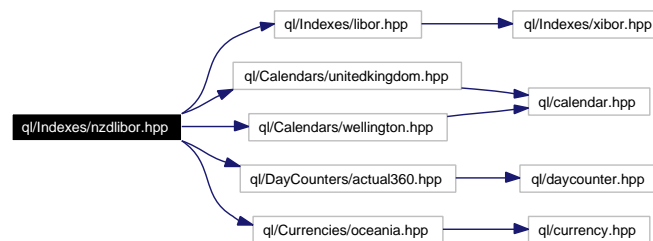
8.106 ql/Indexes/nzdlabor.hpp File Reference

8.106.1 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 nzdlabor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [NZDLibor](#)
NZD LIBOR rate

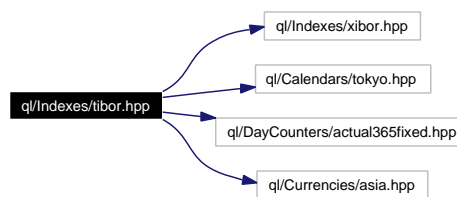
8.107 ql/Indexes/tibor.hpp File Reference

8.107.1 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

8.108 ql/Indexes/trlibor.hpp File Reference

8.108.1 Detailed Description

TRY LIBOR rate

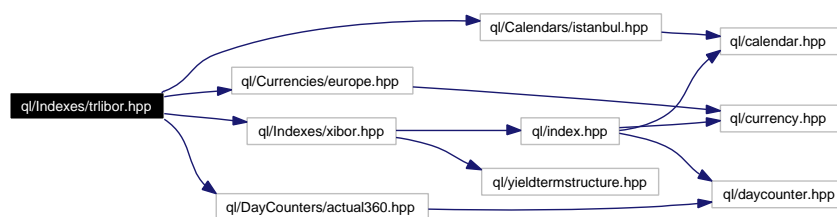
```
#include <ql/Indexes/xibor.hpp>
```

```
#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

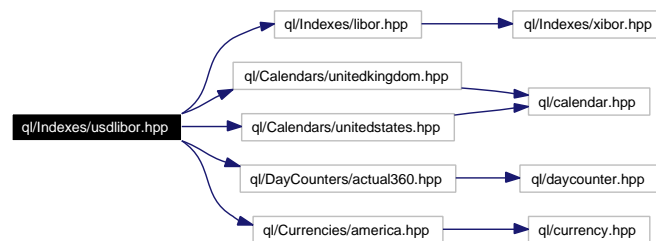
8.109 ql/Indexes/usdlibor.hpp File Reference

8.109.1 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

8.110 ql/Indexes/xibor.hpp File Reference

8.110.1 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

8.111 ql/Indexes/zibor.hpp File Reference

8.111.1 Detailed Description

CHF ZIBOR rate

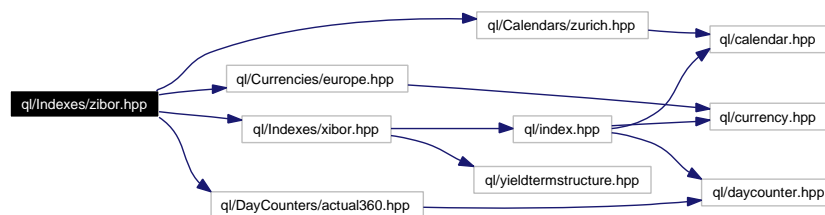
```
#include <ql/Indexes/zibor.hpp>
```

```
#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

8.112 ql/instrument.hpp File Reference

8.112.1 Detailed Description

Abstract instrument class.

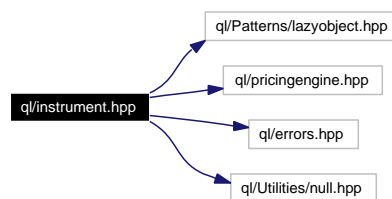
```
#include <ql/Patterns/lazyobject.hpp>
```

```
#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

8.113 ql/Instruments/asianoption.hpp File Reference

8.113.1 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.

8.114 ql/Instruments/barrieroption.hpp File Reference

8.114.1 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

8.115 ql/Instruments/basketoption.hpp File Reference

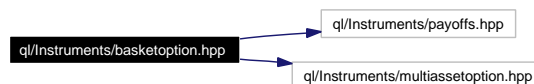
8.115.1 Detailed Description

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

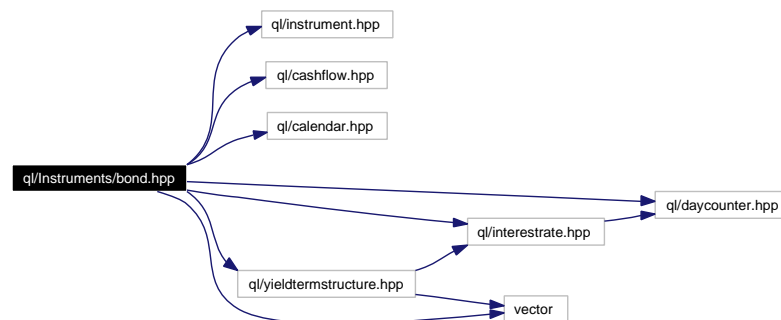
8.116 ql/Instruments/bond.hpp File Reference

8.116.1 Detailed Description

concrete bond class

```
#include <ql/instrument.hpp>
#include <ql/cashflow.hpp>
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/interestrates.hpp>
#include <ql/yieldtermstructure.hpp>
#include <vector>
```

Include dependency graph for bond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Bond**
Base bond class.

8.117 ql/Instruments/callabilityschedule.hpp File Reference

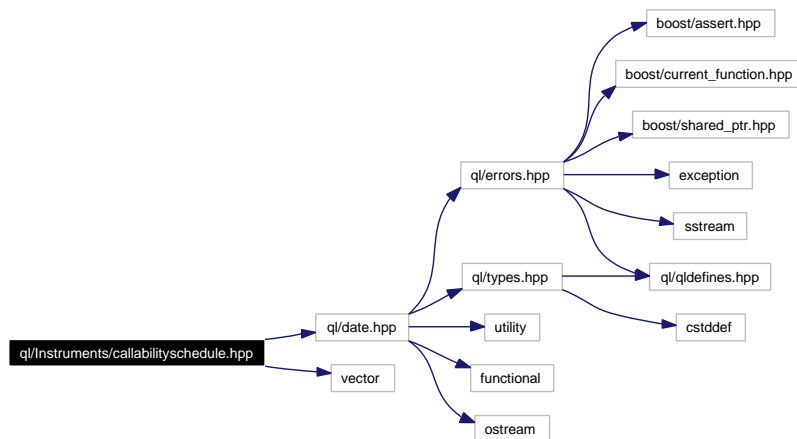
8.117.1 Detailed Description

Schedule of put/call dates.

```
#include <ql/date.hpp>
```

```
#include <vector>
```

Include dependency graph for callabilityschedule.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef `std::vector< Callability >` **QuantLib::CallabilitySchedule**

8.118 ql/Instruments/capfloor.hpp File Reference

8.118.1 Detailed Description

Cap and Floor class.

```
#include <ql/numericalmethod.hpp>
```

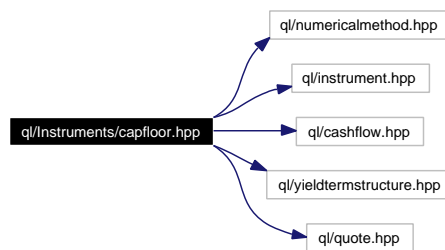
```
#include <ql/instrument.hpp>
```

```
#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

8.119 ql/Instruments/cliquestoption.hpp File Reference

8.119.1 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.

8.120 ql/Instruments/convertiblebond.hpp File Reference

8.120.1 Detailed Description

convertible bond

This is a class under active development. It may change radically. Please subscribe to the quantlib list.

```
#include <ql/Instruments/bond.hpp>
```

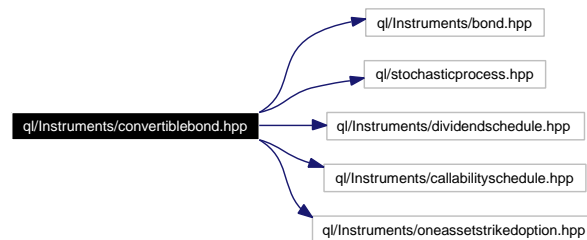
```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/Instruments/dividendschedule.hpp>
```

```
#include <ql/Instruments/callabilityschedule.hpp>
```

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for convertiblebond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ConvertibleBond::option](#)
Option like features for Convertible Bond calculation.
- class [ConvertibleBond::option::arguments](#)
Arguments for Convertible Bond calculation
- class [ConvertibleBond::option::engine](#)
convertible bond engine base class

8.121 ql/Instruments/dividendschedule.hpp File Reference

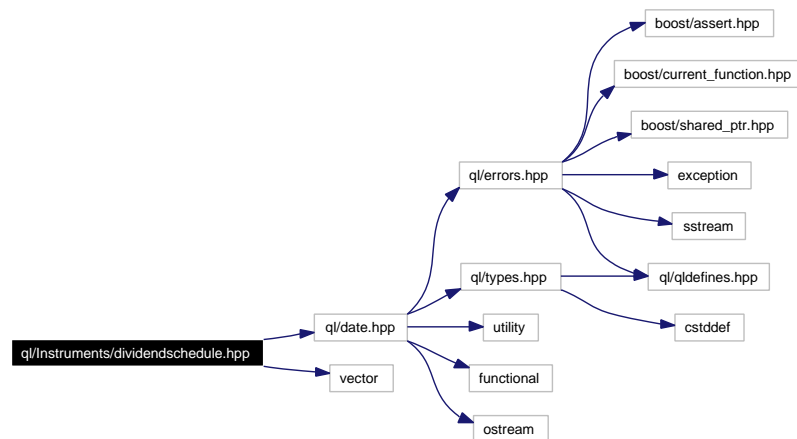
8.121.1 Detailed Description

Schedule of dividend dates.

```
#include <ql/date.hpp>
```

```
#include <vector>
```

Include dependency graph for dividendschedule.hpp:



Namespaces

- namespace **QuantLib**

8.122 ql/Instruments/dividendvanillaoption.hpp File Reference

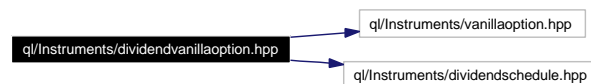
8.122.1 Detailed Description

Vanilla option on a single asset with discrete dividends.

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/Instruments/dividendschedule.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.

8.123 ql/Instruments/europeanoption.hpp File Reference

8.123.1 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.

8.124 ql/Instruments/fixedcouponbond.hpp File Reference

8.124.1 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

8.125 ql/Instruments/floatingratebond.hpp File Reference

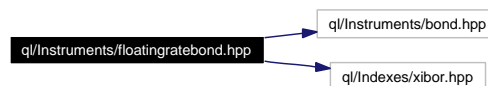
8.125.1 Detailed Description

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

8.126 ql/Instruments/forwardvanillaoption.hpp File Reference

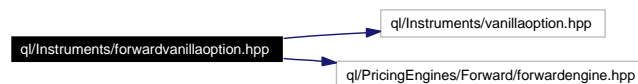
8.126.1 Detailed Description

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.

8.127 ql/Instruments/multiassetoption.hpp File Reference

8.127.1 Detailed Description

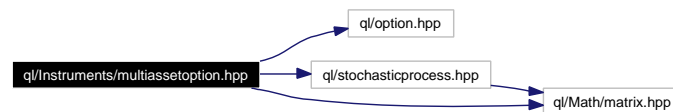
Option on multiple assets.

```
#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

8.128 ql/Instruments/oneassetoption.hpp File Reference

8.128.1 Detailed Description

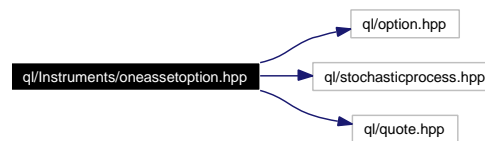
Option on a single asset.

```
#include <ql/option.hpp>
```

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/quote.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

8.129 ql/Instruments/oneassetstrikedoption.hpp File Reference

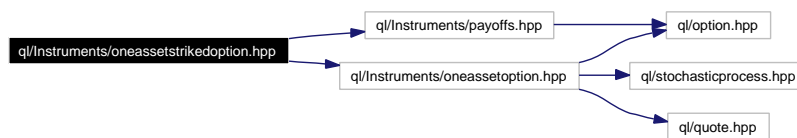
8.129.1 Detailed Description

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.

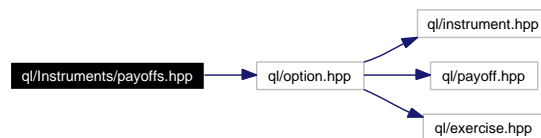
8.130 ql/Instruments/payoffs.hpp File Reference

8.130.1 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.

8.131 ql/Instruments/quantoforwardvanillaoption.hpp File Reference

8.131.1 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.

8.132 ql/Instruments/quantovanillaoption.hpp File Reference

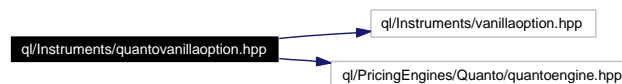
8.132.1 Detailed Description

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

8.133 ql/Instruments/simpleswap.hpp File Reference

8.133.1 Detailed Description

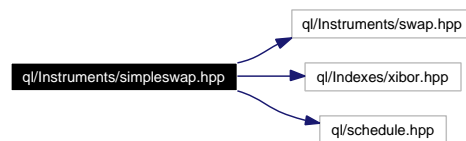
Simple fixed-rate vs Libor swap.

```
#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 [SimpleSwap](#)
Simple fixed-rate vs [Libor](#) swap.
- class [SimpleSwap::arguments](#)
Arguments for simple swap calculation
- class [SimpleSwap::results](#)
Results from simple swap calculation

8.134 ql/Instruments/stock.hpp File Reference

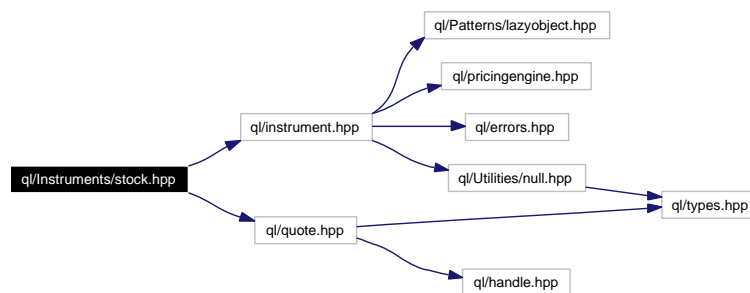
8.134.1 Detailed Description

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.

8.135 ql/Instruments/swap.hpp File Reference

8.135.1 Detailed Description

Interest rate swap.

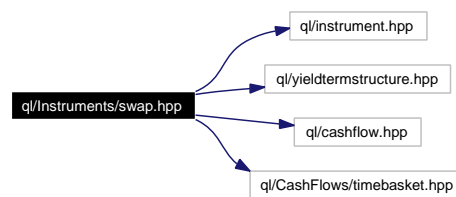
```
#include <ql/instrument.hpp>
```

```
#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.

8.136 ql/Instruments/swaption.hpp File Reference

8.136.1 Detailed Description

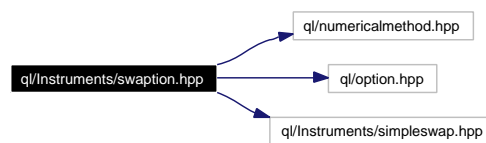
Swaption class.

```
#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

8.137 ql/Instruments/vanillaoption.hpp File Reference

8.137.1 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.

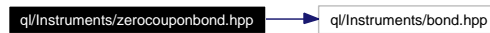
8.138 ql/Instruments/zerocouponbond.hpp File Reference

8.138.1 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

8.139 ql/interestrate.hpp File Reference

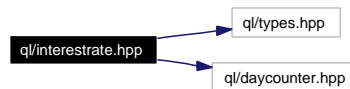
8.139.1 Detailed Description

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.

8.140 ql/Lattices/binomialtree.hpp File Reference

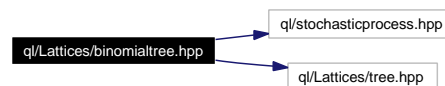
8.140.1 Detailed Description

Binomial tree class.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/Lattices/tree.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.

8.141 ql/Lattices/bsmlattice.hpp File Reference

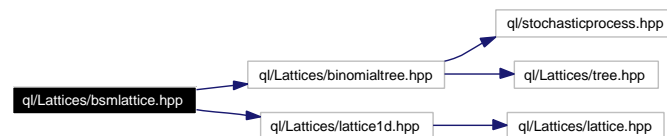
8.141.1 Detailed Description

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.

8.142 ql/Lattices/lattice.hpp File Reference

8.142.1 Detailed Description

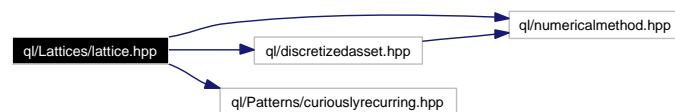
Lattice method class.

```
#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.

8.143 ql/Lattices/lattice1d.hpp File Reference

8.143.1 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.

8.144 ql/Lattices/lattice2d.hpp File Reference

8.144.1 Detailed Description

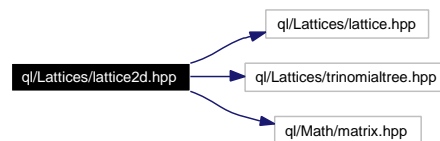
Two-dimensional lattice class.

```
#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.

8.145 ql/Lattices/tree.hpp File Reference

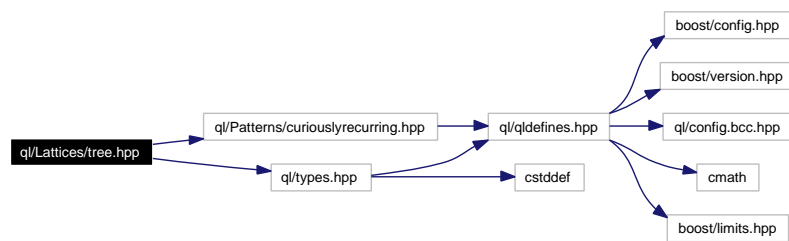
8.145.1 Detailed Description

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

8.146 ql/Lattices/trinomialtree.hpp File Reference

8.146.1 Detailed Description

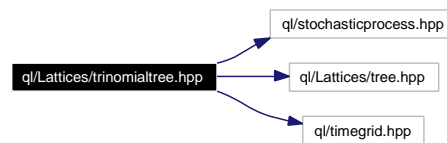
Trinomial tree class.

```
#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.

8.147 ql/Math/array.hpp File Reference

8.147.1 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>
```

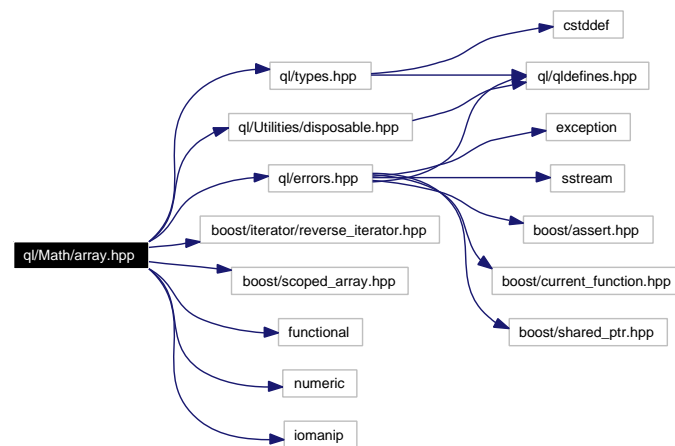
```
#include <boost/scoped_array.hpp>
```

```
#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.

8.148 ql/Math/backwardflatinterpolation.hpp File Reference

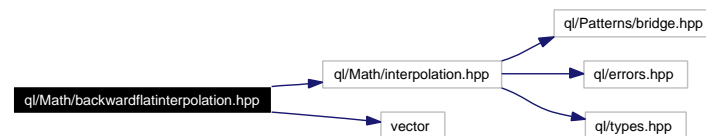
8.148.1 Detailed Description

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.

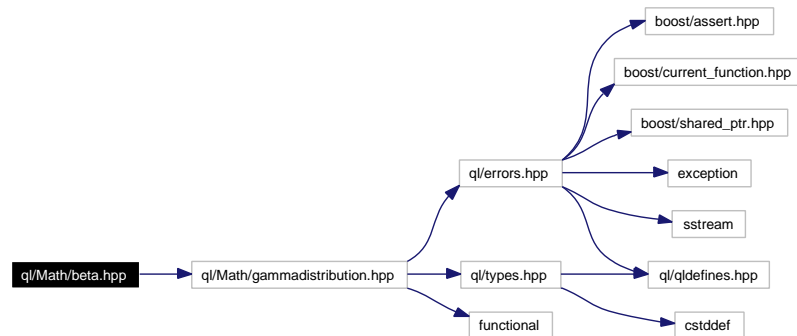
8.149 ql/Math/beta.hpp File Reference

8.149.1 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.

8.150 ql/Math/bicubicsplineinterpolation.hpp File Reference

8.150.1 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

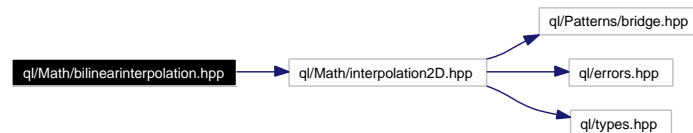
8.151 ql/Math/bilinearinterpolation.hpp File Reference

8.151.1 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

8.152 ql/Math/binomialdistribution.hpp File Reference

8.152.1 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)

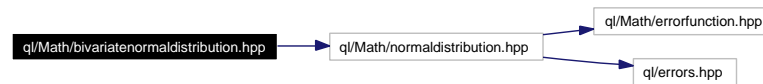
8.153 ql/Math/bivariatenormaldistribution.hpp File Reference

8.153.1 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

8.154 ql/Math/chisquairedistribution.hpp File Reference

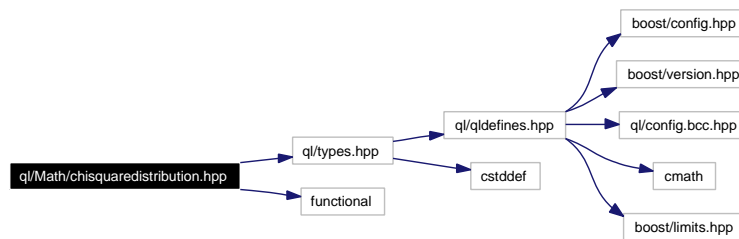
8.154.1 Detailed Description

Chi-square (central and non-central) distributions.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for chisquairedistribution.hpp:



Namespaces

- namespace **QuantLib**

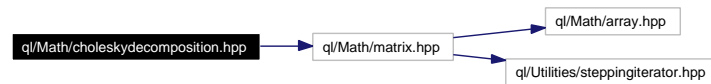
8.155 ql/Math/choleskydecomposition.hpp File Reference

8.155.1 Detailed Description

Cholesky decomposition.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for choleskydecomposition.hpp:



Namespaces

- namespace **QuantLib**

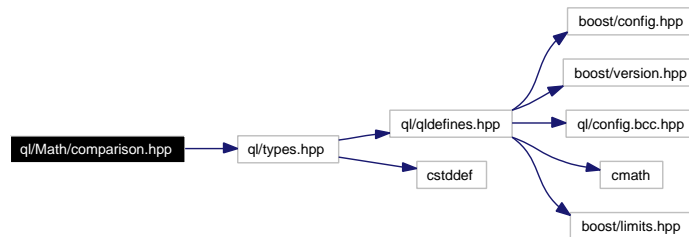
8.156 ql/Math/comparison.hpp File Reference

8.156.1 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)

8.157 ql/Math/convergencestatistics.hpp File Reference

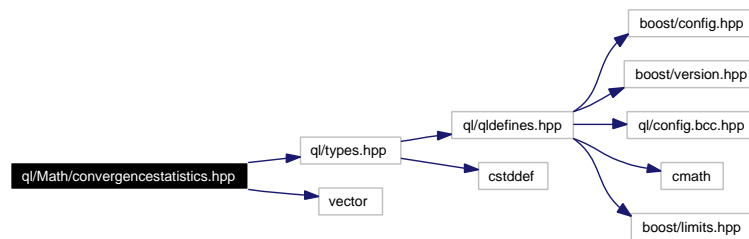
8.157.1 Detailed Description

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

8.158 ql/Math/cubicspline.hpp File Reference

8.158.1 Detailed Description

cubic spline interpolation between discrete points

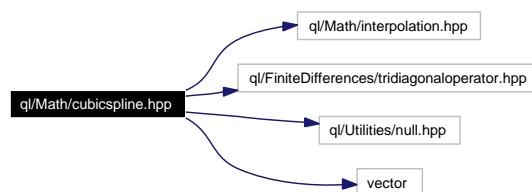
```
#include <ql/Math/interpolation.hpp>
```

```
#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

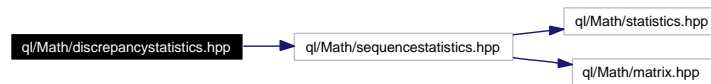
8.159 ql/Math/discrepancystatistics.hpp File Reference

8.159.1 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.

8.160 ql/Math/errorfunction.hpp File Reference

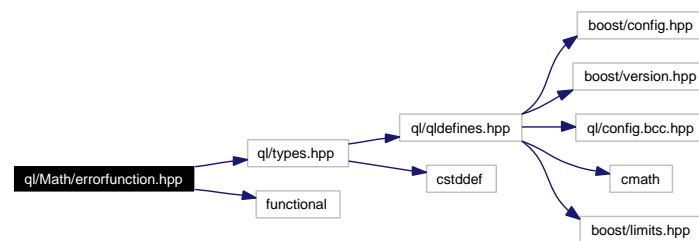
8.160.1 Detailed Description

Error function.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for errorfunction.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ErrorFunction](#)
Error function

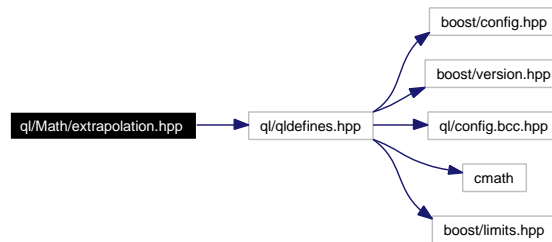
8.161 ql/Math/extrapolation.hpp File Reference

8.161.1 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

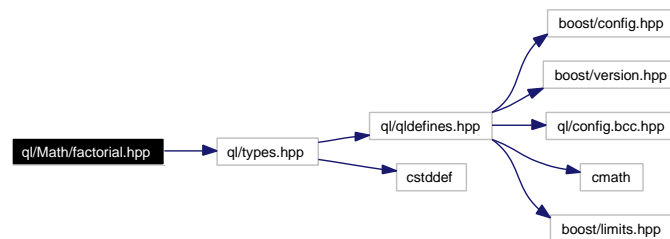
8.162 ql/Math/factorial.hpp File Reference

8.162.1 Detailed Description

Factorial numbers calculator.

```
#include <ql/types.hpp>
```

Include dependency graph for factorial.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Factorial](#)
Factorial numbers calculator

8.163 ql/Math/forwardflatinterpolation.hpp File Reference

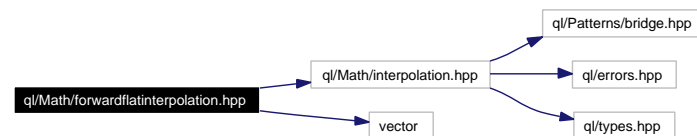
8.163.1 Detailed Description

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.

8.164 ql/Math/functional.hpp File Reference

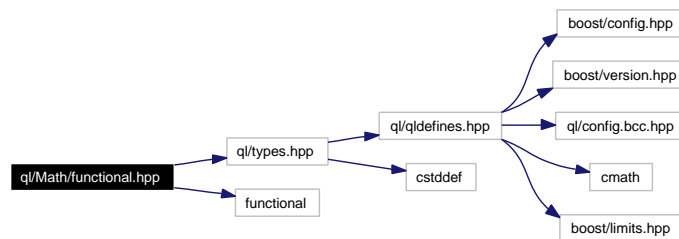
8.164.1 Detailed Description

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)`

8.165 ql/Math/gammadistribution.hpp File Reference

8.165.1 Detailed Description

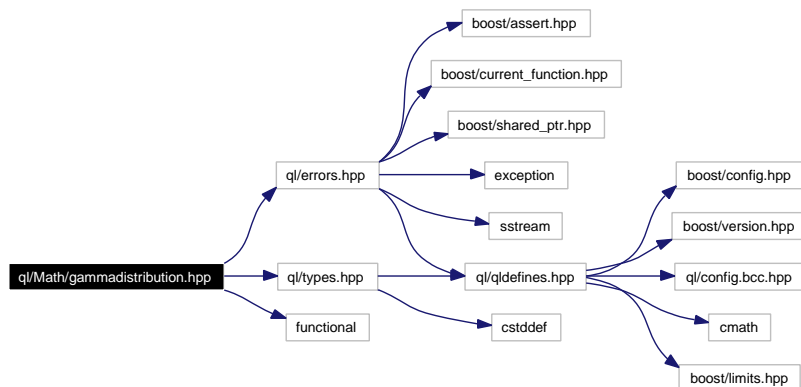
Gamma distribution.

```
#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.

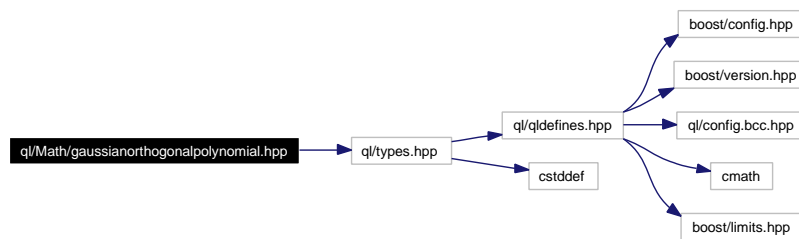
8.166 ql/Math/gaussianorthogonalpolynomial.hpp File Reference

8.166.1 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.

8.167 ql/Math/gaussianquadratures.hpp File Reference

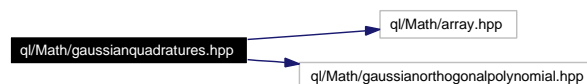
8.167.1 Detailed Description

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

8.168 ql/Math/gaussianstatistics.hpp File Reference

8.168.1 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.

8.169 ql/Math/generalstatistics.hpp File Reference

8.169.1 Detailed Description

statistics tool

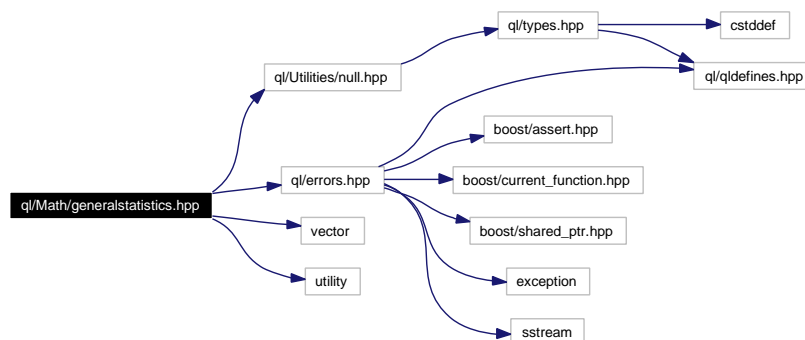
```
#include <ql/Utilities/null.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <vector>
```

```
#include <utility>
```

Include dependency graph for generalstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **GeneralStatistics**
Statistics tool.

8.170 ql/Math/incompletegamma.hpp File Reference

8.170.1 Detailed Description

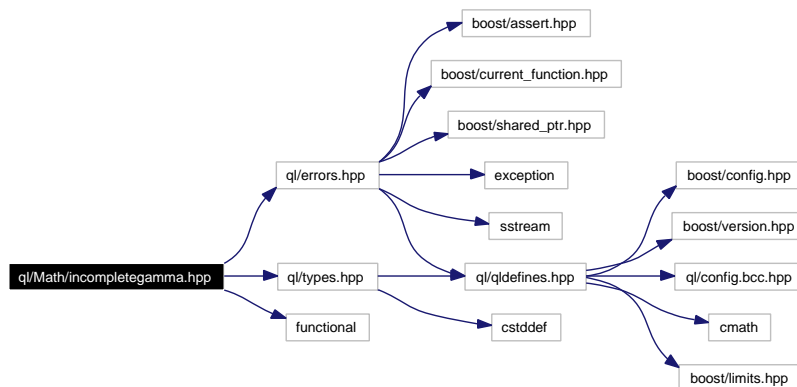
Incomplete Gamma function.

```
#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)

8.171 ql/Math/incrementalstatistics.hpp File Reference

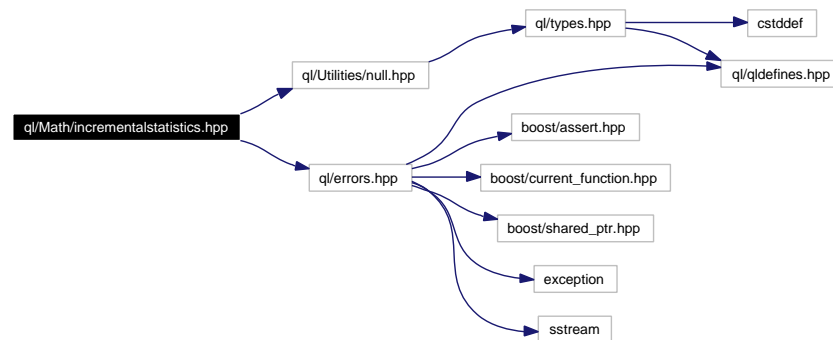
8.171.1 Detailed Description

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.

8.172 ql/Math/interpolation.hpp File Reference

8.172.1 Detailed Description

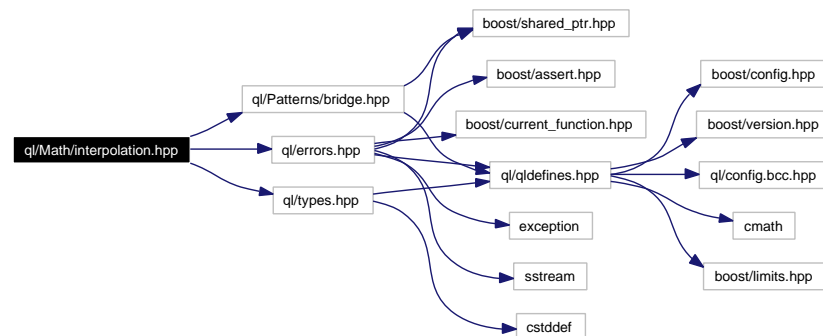
base class for 1-D interpolations

```
#include <ql/Patterns/bridge.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

8.173 ql/Math/interpolation2D.hpp File Reference

8.173.1 Detailed Description

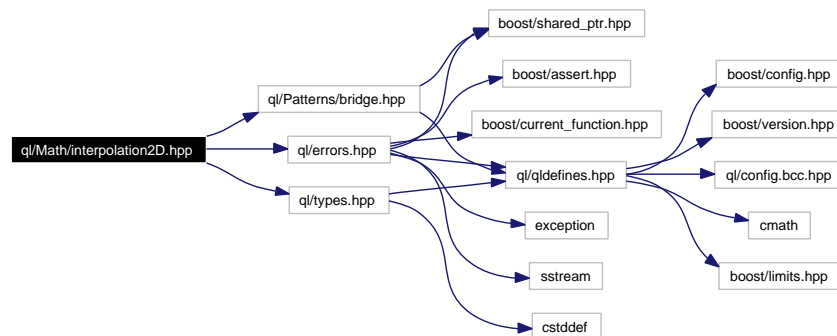
abstract base classes for 2-D interpolations

```
#include <ql/Patterns/bridge.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

8.174 ql/Math/kronrodintegral.hpp File Reference

8.174.1 Detailed Description

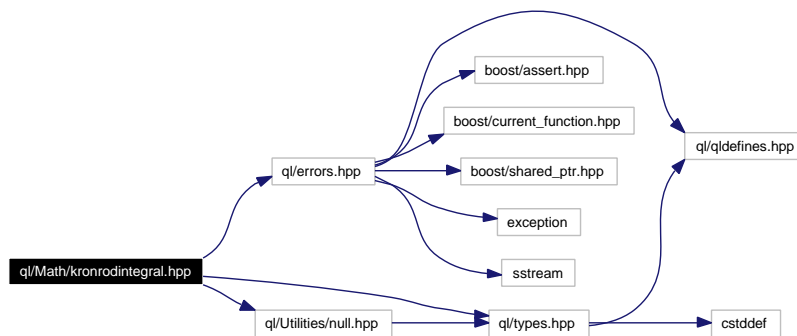
Integral of a 1-dimensional function using the Gauss-Kronrod method.

```
#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.

8.175 ql/Math/lexicographicalview.hpp File Reference

8.175.1 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.

8.176 ql/Math/linearinterpolation.hpp File Reference

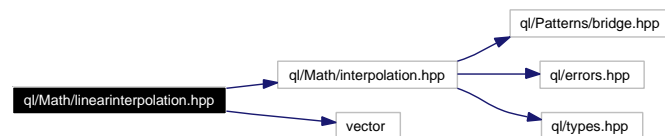
8.176.1 Detailed Description

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.

8.177 ql/Math/loglinearinterpolation.hpp File Reference

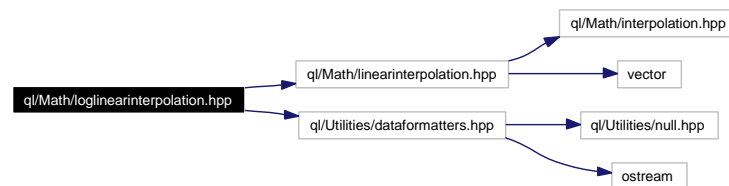
8.177.1 Detailed Description

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

8.178 ql/Math/matrix.hpp File Reference

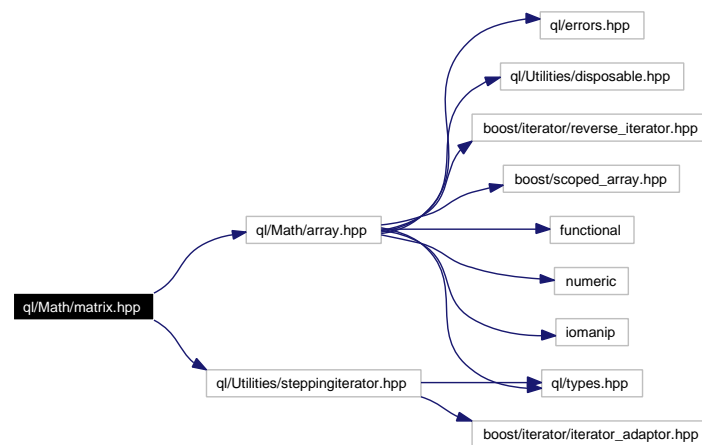
8.178.1 Detailed Description

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.

8.179 ql/Math/multicubicspline.hpp File Reference

8.179.1 Detailed Description

N-dimensional cubic spline interpolation between discrete points.

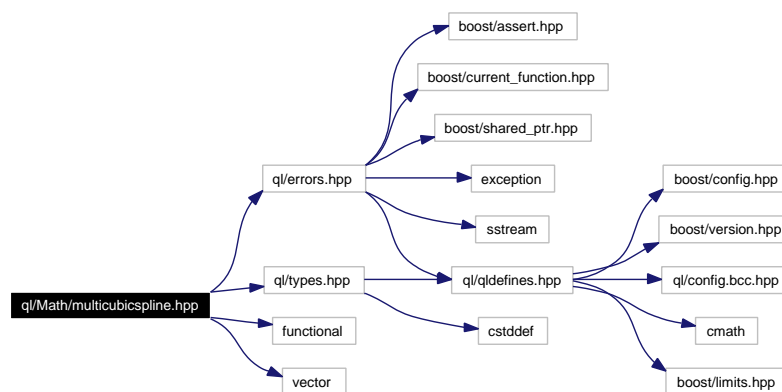
```
#include <ql/errors.hpp>
```

```
#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**

8.180 ql/Math/normaldistribution.hpp File Reference

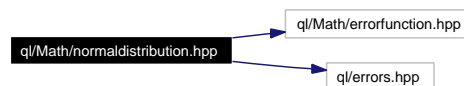
8.180.1 Detailed Description

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**

8.181 ql/Math/poissondistribution.hpp File Reference

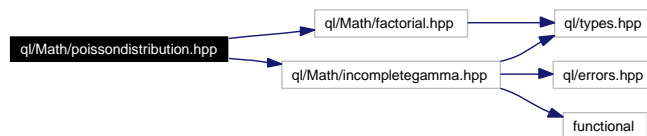
8.181.1 Detailed Description

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.

8.182 ql/Math/primenumbers.hpp File Reference

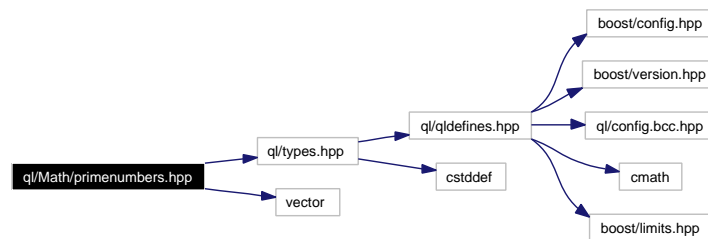
8.182.1 Detailed Description

Prime numbers calculator.

```
#include <ql/types.hpp>
```

```
#include <vector>
```

Include dependency graph for primenumbers.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PrimeNumbers](#)
Prime numbers calculator.

8.183 ql/Math/pseudosqrt.hpp File Reference

8.183.1 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

8.184 ql/Math/riskstatistics.hpp File Reference

8.184.1 Detailed Description

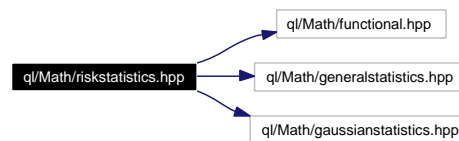
empirical-distribution risk measures

```
#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

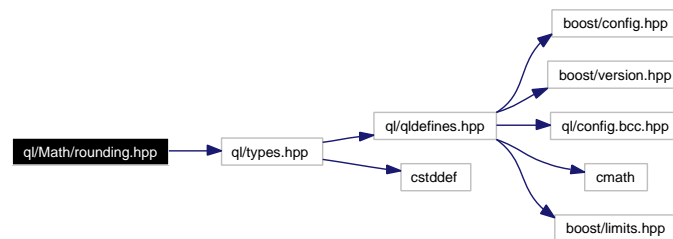
8.185 ql/Math/rounding.hpp File Reference

8.185.1 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.

8.186 ql/Math/sampledcurve.hpp File Reference

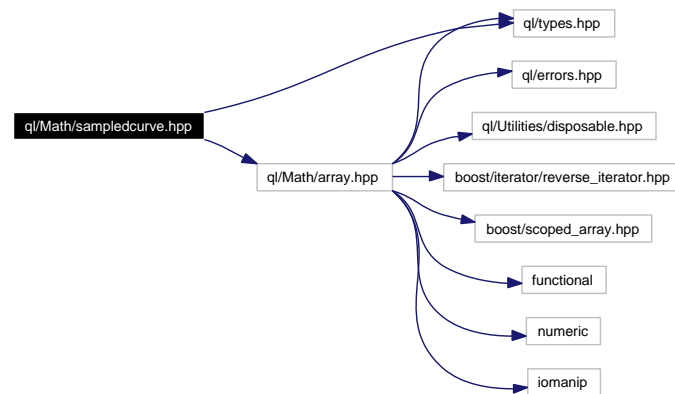
8.186.1 Detailed Description

a class that contains a sampled curve

```
#include <ql/Math/array.hpp>
```

```
#include <ql/types.hpp>
```

Include dependency graph for sampledcurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SampledCurve](#)
This class contains a sampled curve.

Typedefs

- typedef `SampledCurve` **QuantLib::SampledCurveSet**

8.187 ql/Math/segmentintegral.hpp File Reference

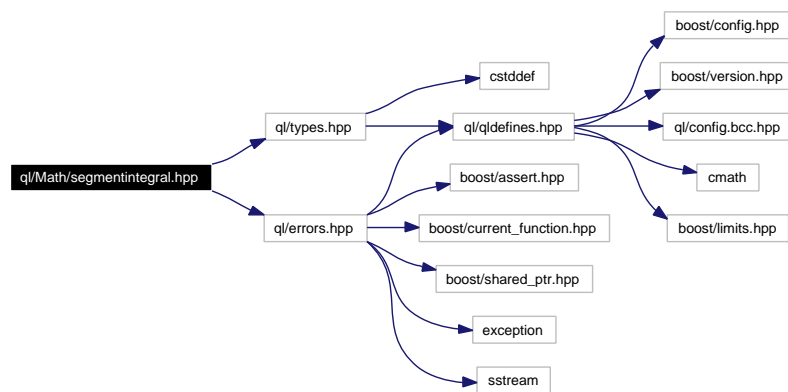
8.187.1 Detailed Description

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.

8.188 ql/Math/sequencestatistics.hpp File Reference

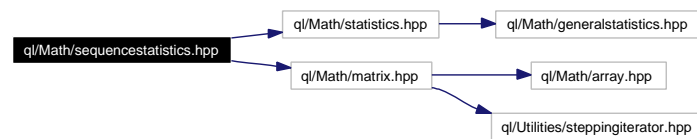
8.188.1 Detailed Description

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)

8.188.2 Define Documentation

8.188.2.1 #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_; \
    }
```

8.188.2.2 #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_; \
    }
```

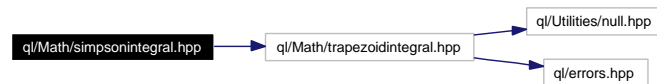
8.189 ql/Math/simpsonintegral.hpp File Reference

8.189.1 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.

8.190 ql/Math/statistics.hpp File Reference

8.190.1 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

8.191 ql/Math/svd.hpp File Reference

8.191.1 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.

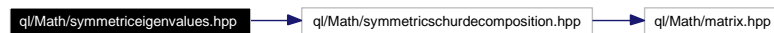
8.192 ql/Math/symmetriceigenvalues.hpp File Reference

8.192.1 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)

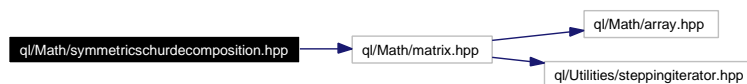
8.193 ql/Math/symmetricschurdecomposition.hpp File Reference

8.193.1 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.

8.194 ql/Math/tqreigendecomposition.hpp File Reference

8.194.1 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

8.195 ql/Math/trapezoidintegral.hpp File Reference

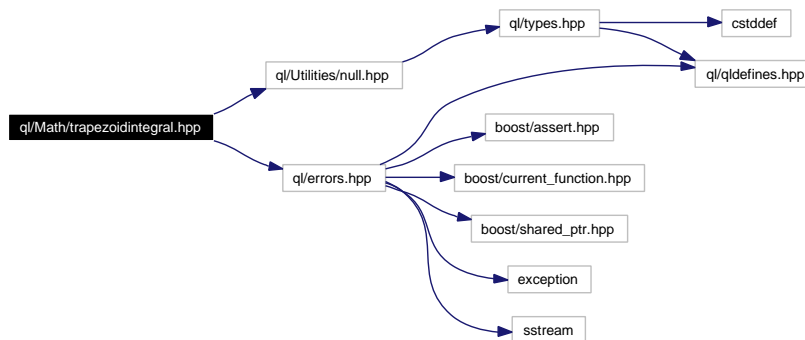
8.195.1 Detailed Description

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.

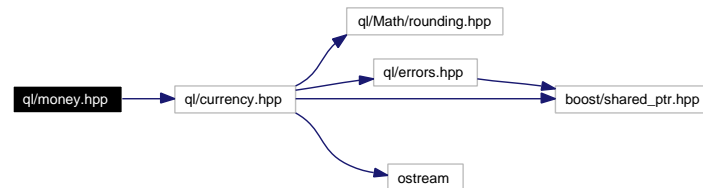
8.196 ql/money.hpp File Reference

8.196.1 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

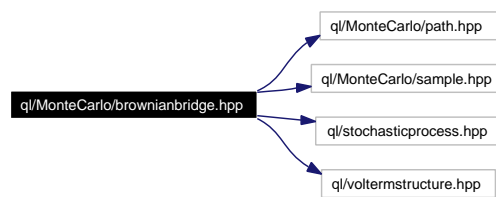
8.197 ql/MonteCarlo/brownianbridge.hpp File Reference

8.197.1 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.

8.198 ql/MonteCarlo/getcovariance.hpp File Reference

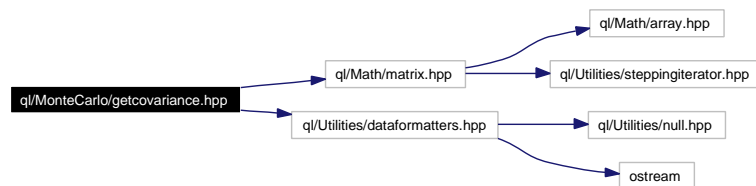
8.198.1 Detailed Description

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)`

8.199 ql/MonteCarlo/mctraits.hpp File Reference

8.199.1 Detailed Description

Monte Carlo policies.

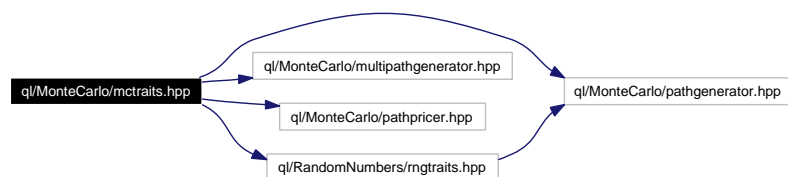
```
#include <ql/MonteCarlo/pathgenerator.hpp>
```

```
#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 [SingleAsset](#)
- struct [MultiVariate](#)
default Monte Carlo traits for multi-variate models
- struct [MultiAsset](#)

8.200 ql/MonteCarlo/mctypedefs.hpp File Reference

8.200.1 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.

8.201 ql/MonteCarlo/montecarlomodel.hpp File Reference

8.201.1 Detailed Description

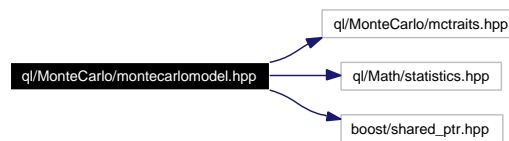
General purpose Monte Carlo model.

```
#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.

8.202 ql/MonteCarlo/multipath.hpp File Reference

8.202.1 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.

8.203 ql/MonteCarlo/multipathgenerator.hpp File Reference

8.203.1 Detailed Description

Generates a multi path from a random-array generator.

```
#include <ql/stochasticprocess.hpp>
```

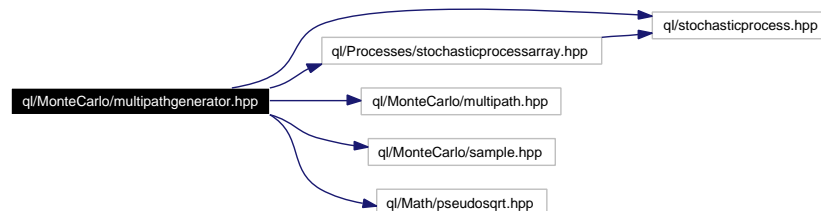
```
#include <ql/Processes/stochasticprocessarray.hpp>
```

```
#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.

8.204 ql/MonteCarlo/path.hpp File Reference

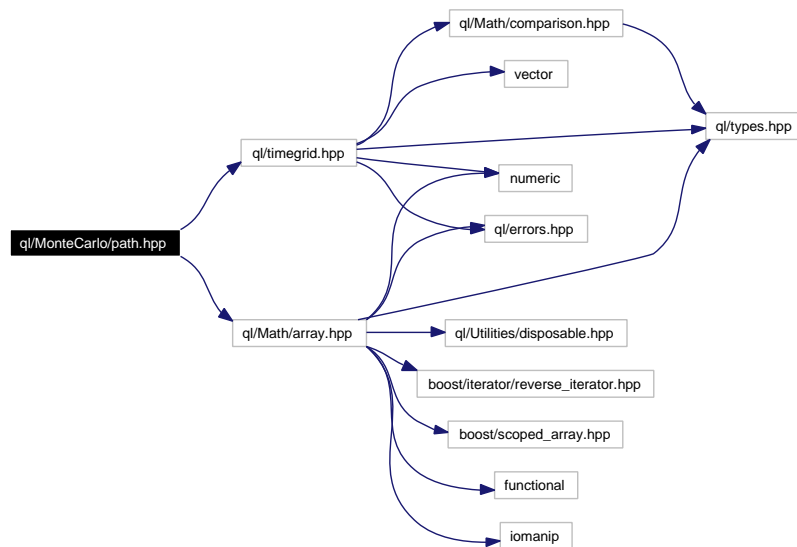
8.204.1 Detailed Description

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`

8.205 ql/MonteCarlo/pathgenerator.hpp File Reference

8.205.1 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.

8.206 ql/MonteCarlo/pathpricer.hpp File Reference

8.206.1 Detailed Description

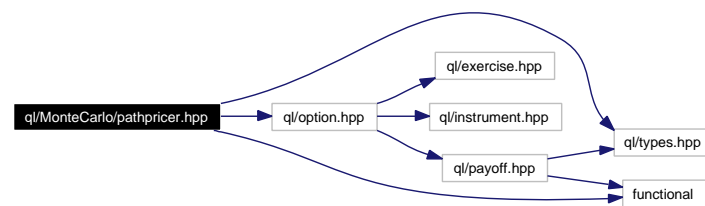
base class for single-path pricers

```
#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

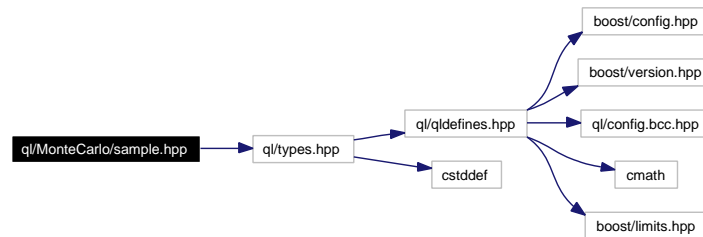
8.207 ql/MonteCarlo/sample.hpp File Reference

8.207.1 Detailed Description

weighted sample

```
#include <ql/types.hpp>
```

Include dependency graph for sample.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct [Sample](#)
weighted sample

8.208 ql/numericalmethod.hpp File Reference

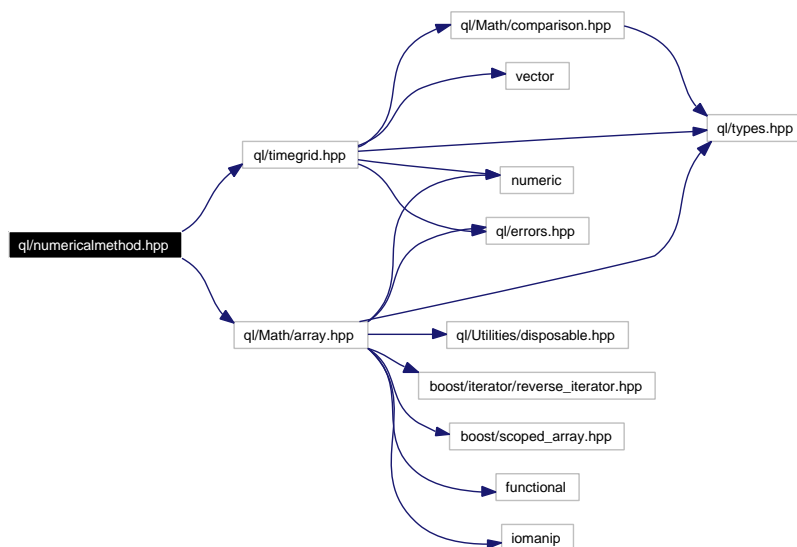
8.208.1 Detailed Description

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.

8.209 ql/Optimization/armijo.hpp File Reference

8.209.1 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.

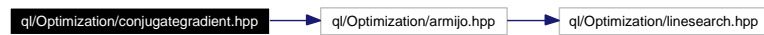
8.210 ql/Optimization/conjugategradient.hpp File Reference

8.210.1 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.

8.211 ql/Optimization/constraint.hpp File Reference

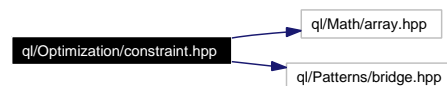
8.211.1 Detailed Description

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

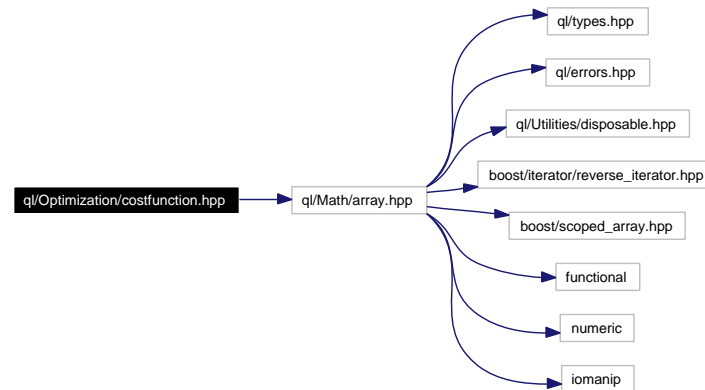
8.212 ql/Optimization/costfunction.hpp File Reference

8.212.1 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.

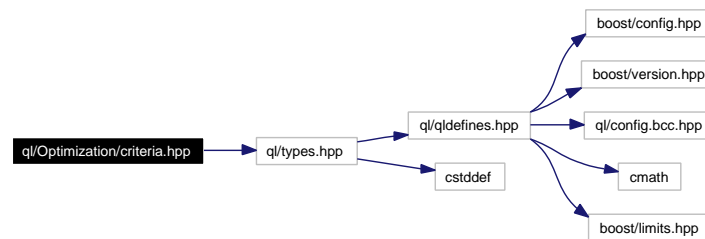
8.213 ql/Optimization/criteria.hpp File Reference

8.213.1 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.

8.214 ql/Optimization/leastsquare.hpp File Reference

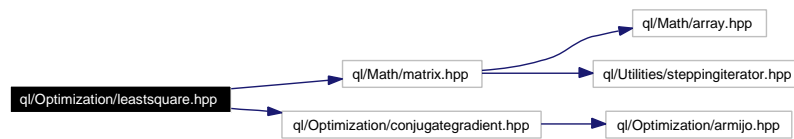
8.214.1 Detailed Description

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.

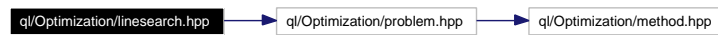
8.215 ql/Optimization/linesearch.hpp File Reference

8.215.1 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.

8.216 ql/Optimization/method.hpp File Reference

8.216.1 Detailed Description

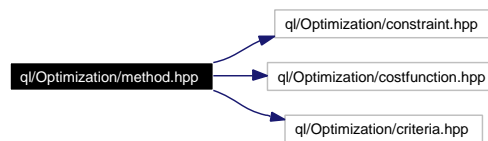
Abstract optimization method class.

```
#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.

8.217 ql/Optimization/problem.hpp File Reference

8.217.1 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.

8.218 ql/Optimization/simplex.hpp File Reference

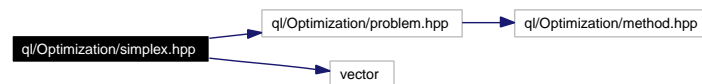
8.218.1 Detailed Description

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.

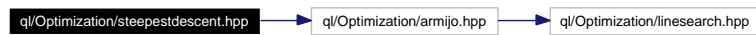
8.219 ql/Optimization/steepestdescent.hpp File Reference

8.219.1 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.

8.220 ql/option.hpp File Reference

8.220.1 Detailed Description

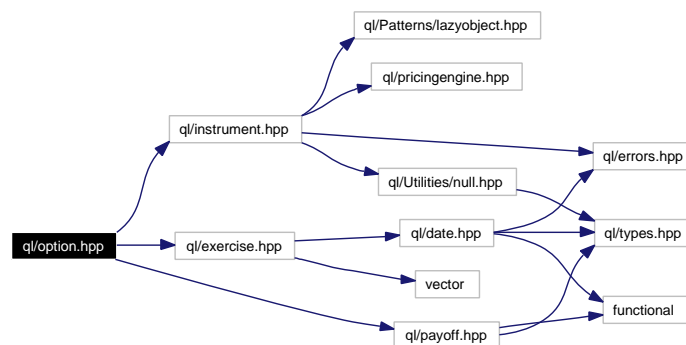
Base option class.

```
#include <ql/instrument.hpp>
```

```
#include <ql/payoff.hpp>
```

```
#include <ql/exercise.hpp>
```

Include dependency graph for option.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Option](#)
base option class
- class [Option::arguments](#)
- class [Greeks](#)
additional option results
- class [MoreGreeks](#)
more additional option results

Functions

- `std::ostream & QuantLib::operator<< (std::ostream &out, Option::Type type)`

8.221 ql/Patterns/bridge.hpp File Reference

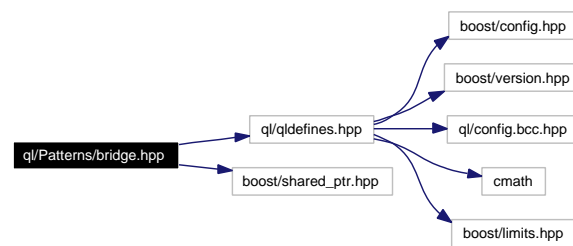
8.221.1 Detailed Description

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.

8.222 ql/Patterns/composite.hpp File Reference

8.222.1 Detailed Description

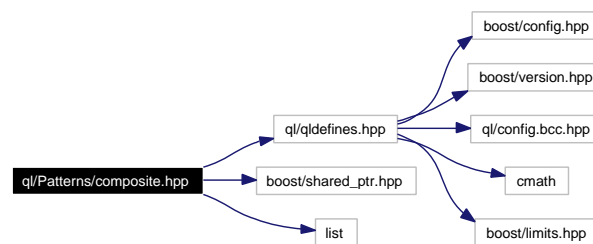
composite pattern

```
#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.

8.223 ql/Patterns/curiouslyrecurring.hpp File Reference

8.223.1 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.

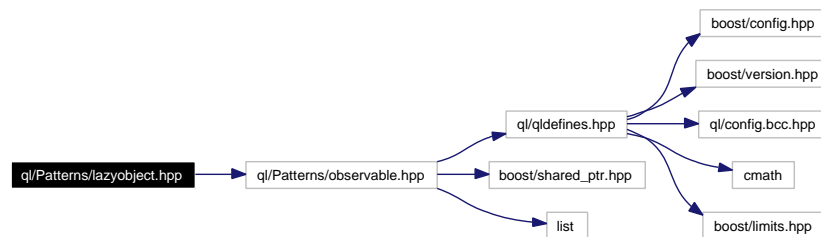
8.224 ql/Patterns/lazyobject.hpp File Reference

8.224.1 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.

8.225 ql/Patterns/observable.hpp File Reference

8.225.1 Detailed Description

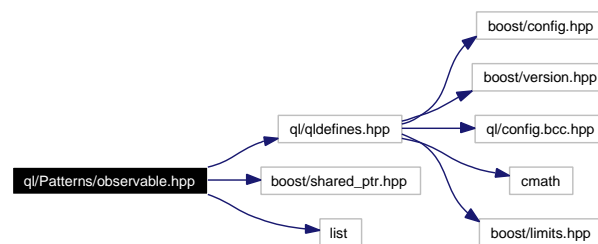
observer/observable pattern

```
#include <ql/qldefines.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.

8.226 ql/Patterns/singleton.hpp File Reference

8.226.1 Detailed Description

basic support for the singleton pattern

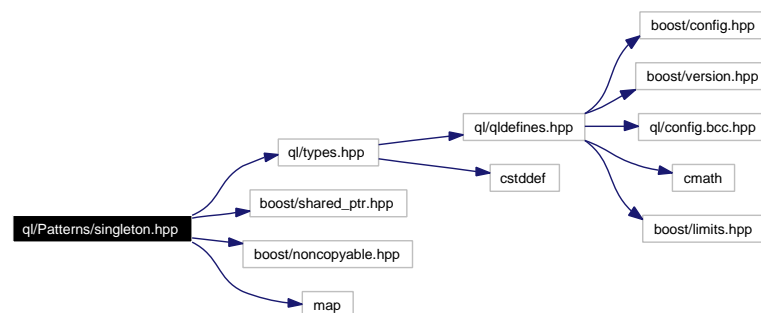
```
#include <ql/types.hpp>
```

```
#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.

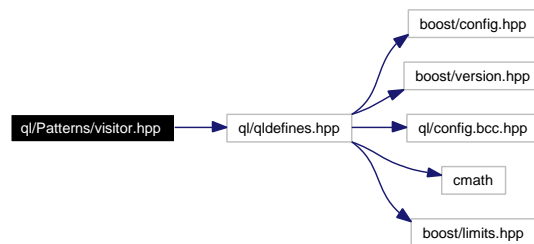
8.227 ql/Patterns/visitor.hpp File Reference

8.227.1 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

8.228 ql/payoff.hpp File Reference

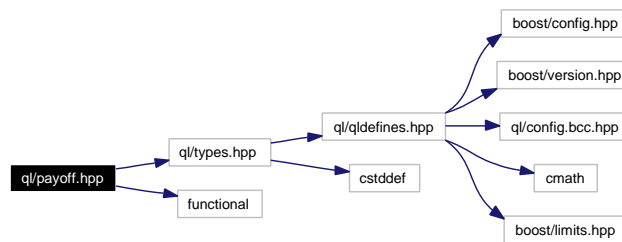
8.228.1 Detailed Description

Option payoff classes.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for payoff.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Payoff**
Base class for option payoffs.

8.229 ql/Pricers/discretegeometricaso.hpp File Reference

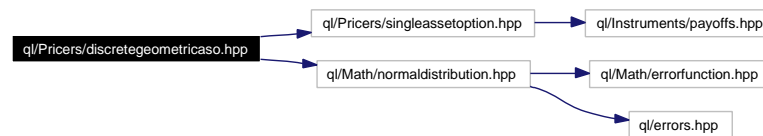
8.229.1 Detailed Description

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).

8.230 ql/Pricers/mccliquestoption.hpp File Reference

8.230.1 Detailed Description

Cliquet option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
```

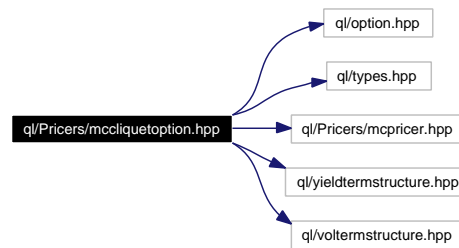
```
#include <ql/types.hpp>
```

```
#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

8.231 ql/Pricers/mcdiscretearithmeticso.hpp File Reference

8.231.1 Detailed Description

Discrete Arithmetic Average Strike Option.

```
#include <ql/option.hpp>
```

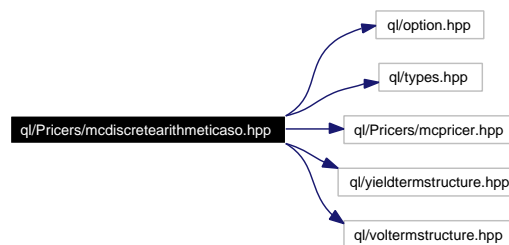
```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for mcdiscretearithmeticso.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [McDiscreteArithmeticASO](#)
example of Monte Carlo pricer using a control variate.

8.232 ql/Pricers/mceverest.hpp File Reference

8.232.1 Detailed Description

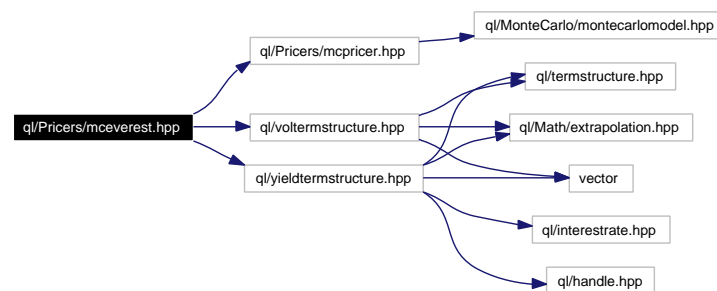
Everest-type option pricer

```
#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.

8.233 ql/Pricers/mchimalaya.hpp File Reference

8.233.1 Detailed Description

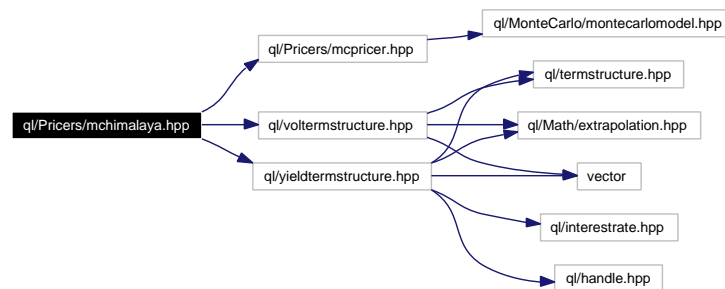
Himalayan-type option pricer.

```
#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.

8.234 ql/Pricers/mcmaxbasket.hpp File Reference

8.234.1 Detailed Description

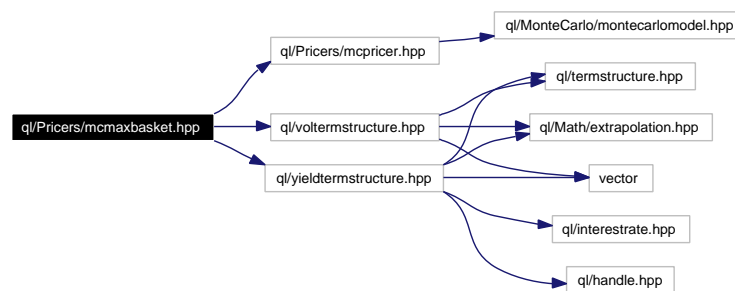
Max Basket Monte Carlo pricer.

```
#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

8.235 ql/Pricers/mcpagoda.hpp File Reference

8.235.1 Detailed Description

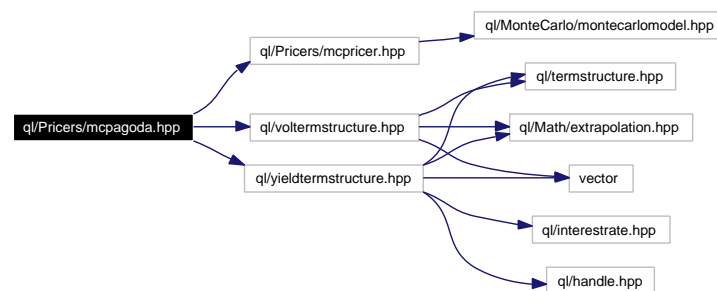
Roofed multi asset Asian option.

```
#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

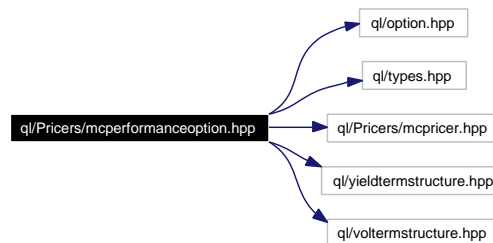
8.236 ql/Pricers/mcperformanceoption.hpp File Reference

8.236.1 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.

8.237 ql/Pricers/mcpricer.hpp File Reference

8.237.1 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

8.238 ql/Pricers/singleassetoption.hpp File Reference

8.238.1 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.

8.239 ql/pricingengine.hpp File Reference

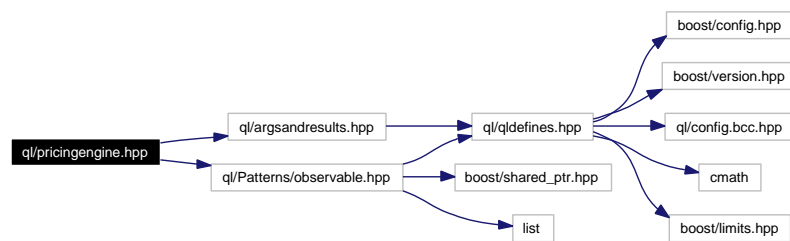
8.239.1 Detailed Description

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

8.240 `ql/PricingEngines/americanpayoffatexpiry.hpp` File Reference

8.240.1 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](#)

8.241 ql/PricingEngines/americanpayoffathit.hpp File Reference

8.241.1 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](#)

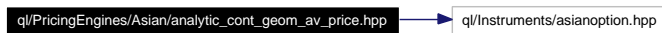
8.242 ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp File Reference

8.242.1 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.

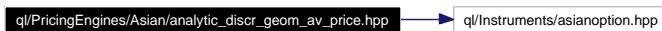
8.243 ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp File Reference

8.243.1 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.

8.244 ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp File Reference

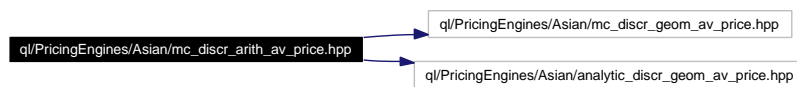
8.244.1 Detailed Description

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.

8.245 ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp File Reference

8.245.1 Detailed Description

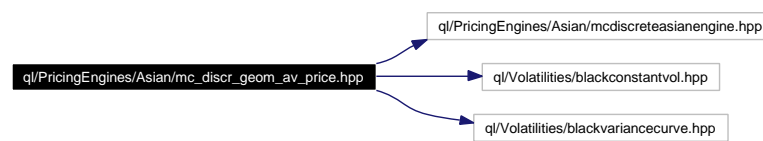
Monte Carlo engine for discrete geometric average price Asian.

```
#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.

8.246 ql/PricingEngines/Asian/mcdiscreteasianengine.hpp File Reference

8.246.1 Detailed Description

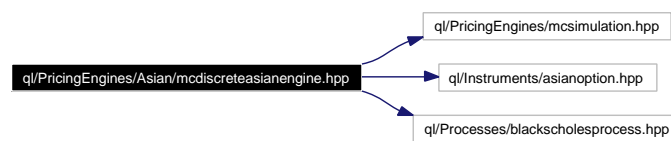
Monte Carlo pricing engine for discrete average Asians.

```
#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.

8.247 ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference

8.247.1 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.

8.248 ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference

8.248.1 Detailed Description

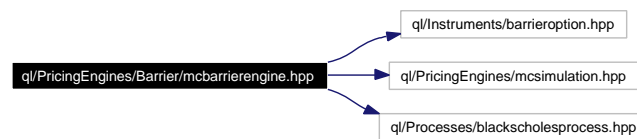
Monte Carlo barrier option engines.

```
#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.

8.249 ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference

8.249.1 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

8.250 ql/PricingEngines/Basket/mcbasketengine.hpp File Reference

8.250.1 Detailed Description

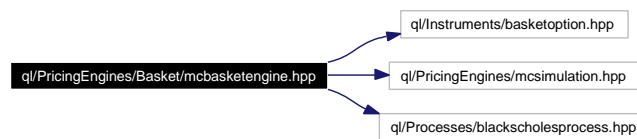
European basket MC Engine.

```
#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.

8.251 ql/PricingEngines/Basket/stulzengine.hpp File Reference

8.251.1 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.

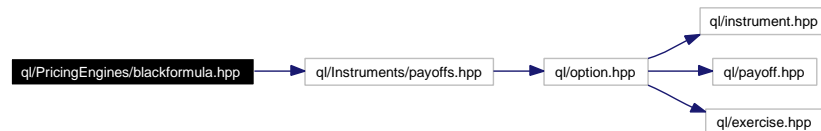
8.252 ql/PricingEngines/blackformula.hpp File Reference

8.252.1 Detailed Description

Black formula.

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for blackformula.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BlackFormula](#)
Black-formula calculator.

8.253 ql/PricingEngines/blackmodel.hpp File Reference

8.253.1 Detailed Description

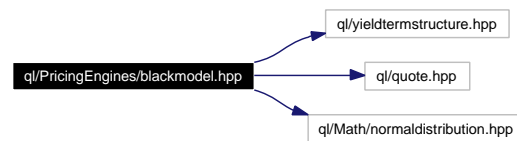
Abstract class for Black-type models (market models).

```
#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.

8.254 ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference

8.254.1 Detailed Description

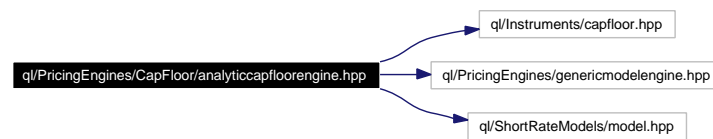
Analytic engine for caps/floors.

```
#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.

8.255 ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference

8.255.1 Detailed Description

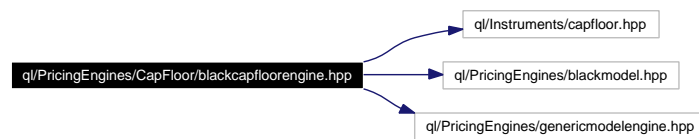
Black-formula cap/floor engine.

```
#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.

8.256 ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference

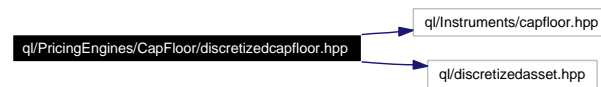
8.256.1 Detailed Description

discretized cap/floor

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for discretizedcapfloor.hpp:



Namespaces

- namespace **QuantLib**

8.257 ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference

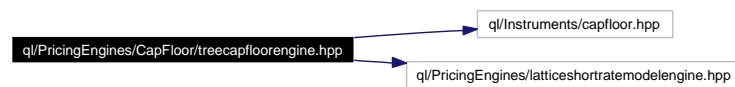
8.257.1 Detailed Description

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.

8.258 ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference

8.258.1 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.

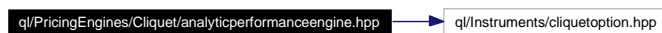
8.259 ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference

8.259.1 Detailed Description

Analytic performance engine.

```
#include <ql/Instruments/cliquestoption.hpp>
```

Include dependency graph for analyticperformanceengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticPerformanceEngine](#)
Pricing engine for performance options using analytical formulae.

8.260 `ql/PricingEngines/Cliquet/mccliquetengine.hpp` File Reference

8.260.1 Detailed Description

Monte Carlo Cliquet option engine.

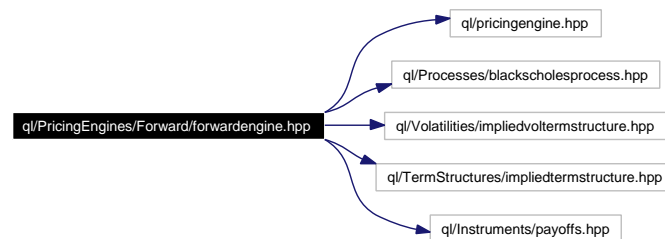
8.261 ql/PricingEngines/Forward/forwardengine.hpp File Reference

8.261.1 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.

8.262 ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference

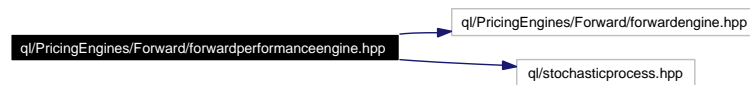
8.262.1 Detailed Description

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.

8.263 ql/PricingEngines/genericmodelengine.hpp File Reference

8.263.1 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.

8.264 ql/PricingEngines/greeks.hpp File Reference

8.264.1 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

8.265 ql/PricingEngines/latticeshortratemodelengine.hpp File Reference

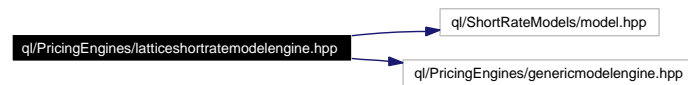
8.265.1 Detailed Description

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.

8.266 ql/PricingEngines/mcsimulation.hpp File Reference

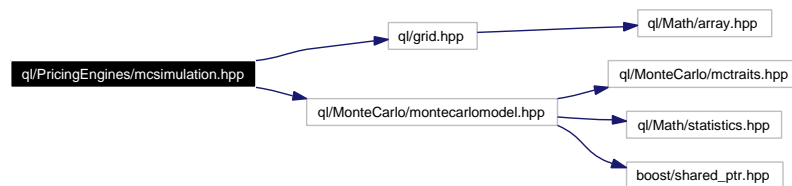
8.266.1 Detailed Description

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

8.267 ql/PricingEngines/Quanto/quantoengine.hpp File Reference

8.267.1 Detailed Description

Quanto option engine.

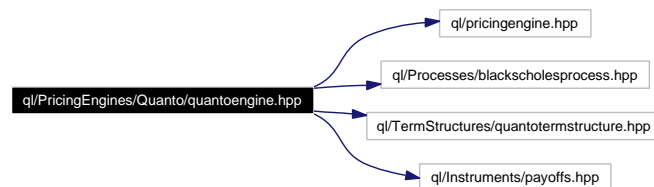
```
#include <ql/pricingengine.hpp>
```

```
#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.

8.268 ql/PricingEngines/Swaption/blackswaptionengine.hpp File Reference

8.268.1 Detailed Description

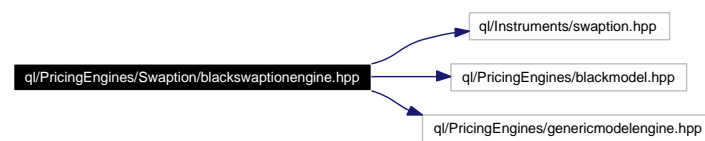
Black-formula swaption engine.

```
#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.

8.269 ql/PricingEngines/Swaption/discretizedswaption.hpp File Reference

8.269.1 Detailed Description

Discretized swaption class.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for discretizedswaption.hpp:



Namespaces

- namespace **QuantLib**

8.270 ql/PricingEngines/Swaption/g2swaptionengine.hpp File Reference

8.270.1 Detailed Description

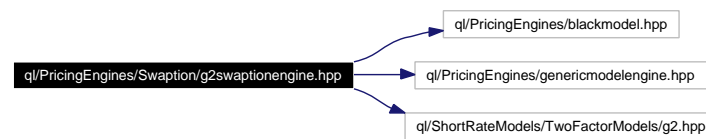
Swaption pricing engine for two-factor additive Gaussian Model G2++.

```
#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

8.271 ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp File Reference

8.271.1 Detailed Description

Swaption engine using Jamshidian's decomposition.

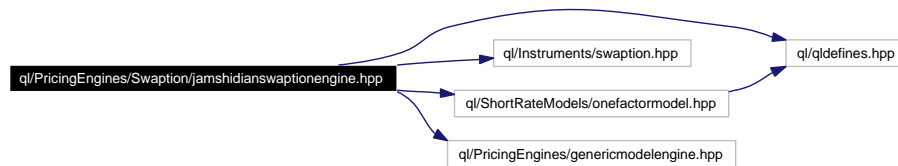
```
#include <ql/qldefines.hpp>
```

```
#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.

8.272 ql/PricingEngines/Swaption/treeswaptionengine.hpp File Reference

8.272.1 Detailed Description

Numerical lattice engine for swaptions.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

Include dependency graph for treeswaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TreeSwaptionEngine](#)
Numerical lattice engine for swaptions.

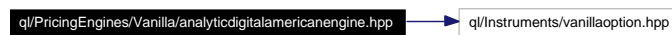
8.273 ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference

8.273.1 Detailed Description

analytic digital American option engine

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for analyticdigitalamericanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticDigitalAmericanEngine](#)

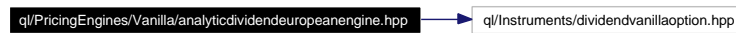
8.274 ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp File Reference

8.274.1 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.

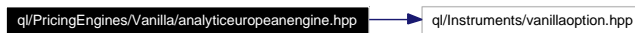
8.275 ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference

8.275.1 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.

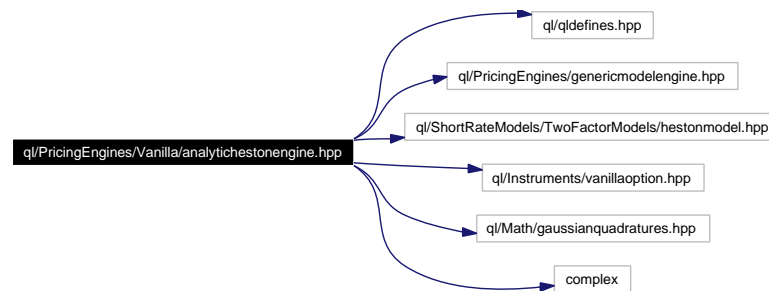
8.276 ql/PricingEngines/Vanilla/analytichestonengine.hpp File Reference

8.276.1 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

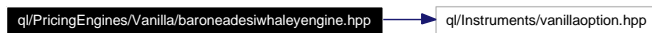
8.277 ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference

8.277.1 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](#)

8.278 ql/PricingEngines/Vanilla/batesengine.hpp File Reference

8.278.1 Detailed Description

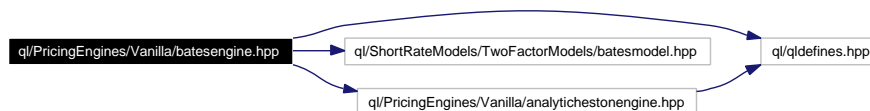
analytic Bates model engine

```
#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.

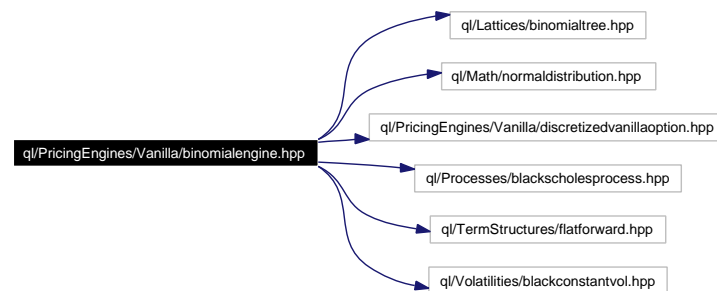
8.279 ql/PricingEngines/Vanilla/binomialengine.hpp File Reference

8.279.1 Detailed Description

Binomial option engine.

```
#include <ql/Lattices/binomialtree.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.

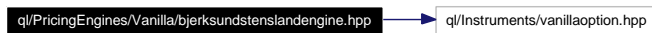
8.280 ql/PricingEngines/Vanilla/bjersundstenslandengine.hpp File Reference

8.280.1 Detailed Description

Bjersund and Stensland approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for bjersundstenslandengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BjersundStenslandApproximationEngine](#)

8.281 ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference

8.281.1 Detailed Description

discretized vanilla option

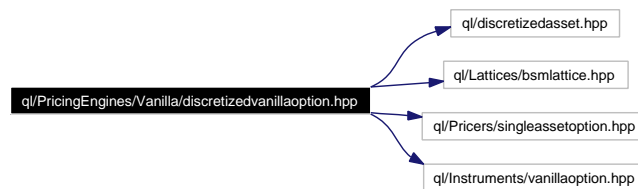
```
#include <ql/discretizedasset.hpp>
```

```
#include <ql/Lattices/bsmlattice.hpp>
```

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for discretizedvanillaoption.hpp:



Namespaces

- namespace **QuantLib**

8.282 `ql/PricingEngines/Vanilla/fdamericanengine.hpp` File Reference

8.282.1 Detailed Description

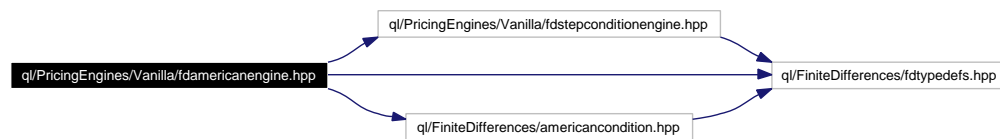
Finite-differences American option engine.

```
#include <ql/PricingEngines/Vanilla/fdstepconditionengine.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/FiniteDifferences/americancondition.hpp>
```

Include dependency graph for `fdamericanengine.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [FDAmericanEngine](#)

Finite-differences pricing engine for American one asset options.

8.283 ql/PricingEngines/Vanilla/fdbermudanengine.hpp File Reference

8.283.1 Detailed Description

finite-difference Bermudan engine

```
#include <ql/PricingEngines/Vanilla/fdmultipleriodengine.hpp>
```

Include dependency graph for fdbermudanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FDBermudanEngine](#)
Finite-differences Bermudan engine.

8.284 ql/PricingEngines/Vanilla/fddividendamericanengine.hpp File Reference

8.284.1 Detailed Description

american engine with discrete deterministic dividends

```
#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>
```

```
#include <ql/FiniteDifferences/americancondition.hpp>
```

Include dependency graph for fddividendamericanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FDDividendAmericanEngine](#)
Finite-differences pricing engine for dividend American options.

8.285 ql/PricingEngines/Vanilla/fddividendengine.hpp File Reference

8.285.1 Detailed Description

base engine for option with dividends

```
#include <ql/PricingEngines/Vanilla/fdmultipleriodengine.hpp>
```

Include dependency graph for fddividendengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FDDividendEngine](#)
Base finite-differences pricing engine for dividend options.

8.286 ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp File Reference

8.286.1 Detailed Description

finite-differences engine for European option with dividends

```
#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>
```

Include dependency graph for fddividendeuropeanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FDDividendEuropeanEngine](#)
Finite-differences pricing engine for dividend European options.

8.287 ql/PricingEngines/Vanilla/fddividendshoutengine.hpp File Reference

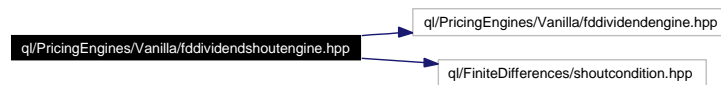
8.287.1 Detailed Description

base class for shout engine with dividends

```
#include <ql/PricingEngines/Vanilla/fddividendengine.hpp>
```

```
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

Include dependency graph for fddividendshoutengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FDDividendShoutEngine](#)
Finite-differences shout engine with dividends.

8.288 ql/PricingEngines/Vanilla/fdeuropeanengine.hpp File Reference

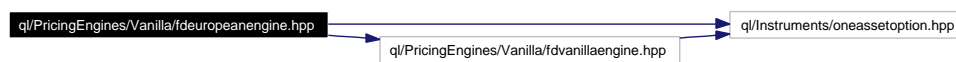
8.288.1 Detailed Description

Finite-difference European engine.

```
#include <ql/Instruments/oneassetoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>
```

Include dependency graph for fdeuropeanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FDEuropeanEngine](#)
Pricing engine for European options using finite-differences.

8.289 ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp File Reference

8.289.1 Detailed Description

base engine for options with events happening at specific times

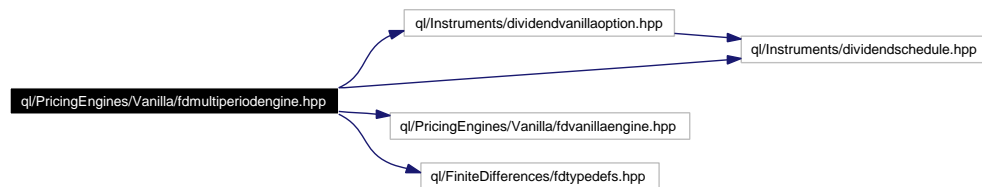
```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

```
#include <ql/PricingEngines/Vanilla/fdvanillaengine.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/Instruments/dividendschedule.hpp>
```

Include dependency graph for fdmultiperiodengine.hpp:



Namespaces

- namespace **QuantLib**

8.290 ql/PricingEngines/Vanilla/fdshoutengine.hpp File Reference

8.290.1 Detailed Description

Finite-differences shout engine.

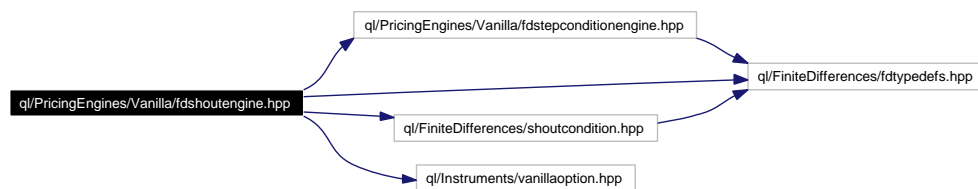
```
#include <ql/PricingEngines/Vanilla/fdstepconditionengine.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for fdshoutengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FDS shoutEngine**

Finite-differences pricing engine for shout vanilla options.

8.291 ql/PricingEngines/Vanilla/fdstepconditionengine.hpp File Reference

8.291.1 Detailed Description

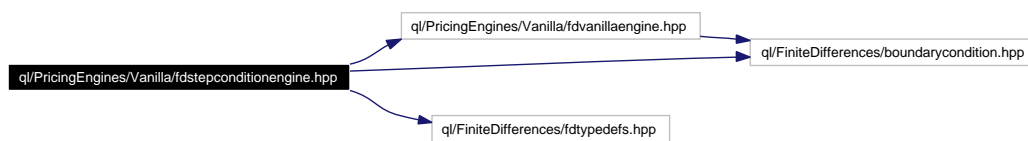
Finite-differences step-condition engine.

```
#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.

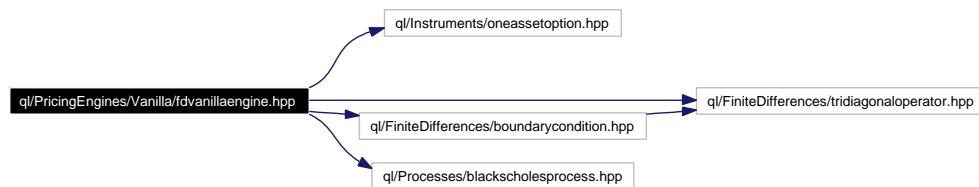
8.292 ql/PricingEngines/Vanilla/fdvanillaengine.hpp File Reference

8.292.1 Detailed Description

Finite-differences vanilla-option engine.

```
#include <ql/Instruments/oneassetoption.hpp>
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
#include <ql/FiniteDifferences/boundarycondition.hpp>
#include <ql/Processes/blackscholesprocess.hpp>
```

Include dependency graph for fdvanillaengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FDVanillaEngine](#)
Finite-differences pricing engine for BSM one asset options.

8.293 ql/PricingEngines/Vanilla/integralengine.hpp File Reference

8.293.1 Detailed Description

Integral option engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for integralengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [IntegralEngine](#)

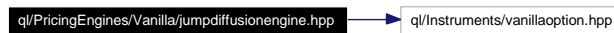
8.294 ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference

8.294.1 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.

8.295 ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference

8.295.1 Detailed Description

Ju quadratic (1999) approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for juquadraticengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [JuQuadraticApproximationEngine](#)

8.296 ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference

8.296.1 Detailed Description

digital option Monte Carlo engine

```
#include <ql/exercise.hpp>
```

```
#include <ql/yieldtermstructure.hpp>
```

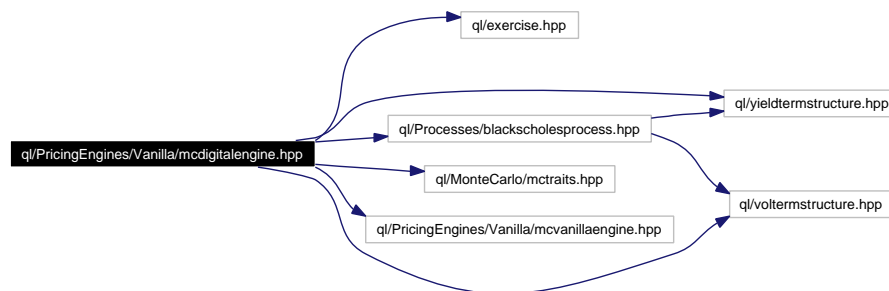
```
#include <ql/voltermstructure.hpp>
```

```
#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.

8.297 ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference

8.297.1 Detailed Description

Monte Carlo European option engine.

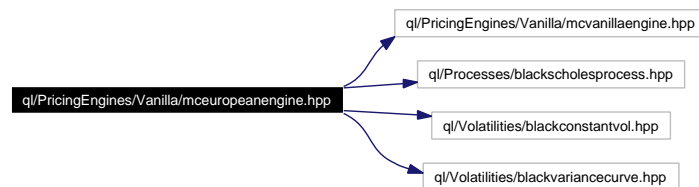
```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

```
#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.

8.298 ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp File Reference

8.298.1 Detailed Description

Monte Carlo Heston-model engine for European options.

```
#include <ql/PricingEngines/Vanilla/mchestonengine.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.

8.299 ql/PricingEngines/Vanilla/mchestonengine.hpp File Reference

8.299.1 Detailed Description

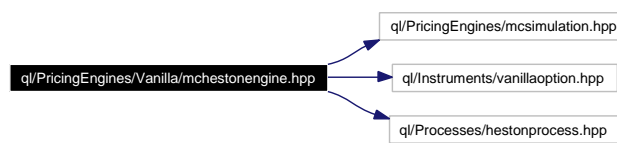
Monte Carlo Heston-model engine.

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/Processes/hestonprocess.hpp>
```

Include dependency graph for mchestonengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCHestonEngine](#)
Monte Carlo Heston-model engine.

8.300 ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference

8.300.1 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.

8.301 ql/Processes/blackscholesprocess.hpp File Reference

8.301.1 Detailed Description

Black-Scholes processes.

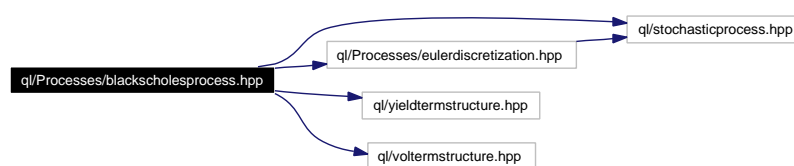
```
#include <ql/stochasticprocess.hpp>
```

```
#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.

8.302 ql/Processes/capletlmmprocess.hpp File Reference

8.302.1 Detailed Description

stochastic process of a (cap) libor market model

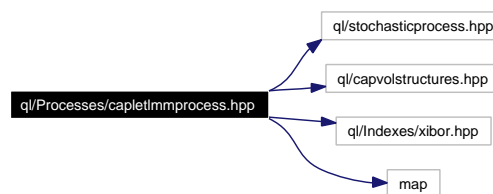
```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <map>
```

Include dependency graph for capletlmmprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CapletLiborMarketModelProcess](#)
caplet libor-market-model process

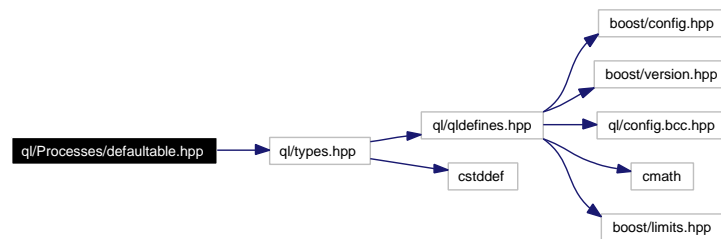
8.303 ql/Processes/defaultable.hpp File Reference

8.303.1 Detailed Description

Defaultable processes.

```
#include <ql/types.hpp>
```

Include dependency graph for defaultable.hpp:



Namespaces

- namespace **QuantLib**

8.304 ql/Processes/eulerdiscretization.hpp File Reference

8.304.1 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.

8.305 ql/Processes/geometricbrownianprocess.hpp File Reference

8.305.1 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.

8.306 ql/Processes/hestonprocess.hpp File Reference

8.306.1 Detailed Description

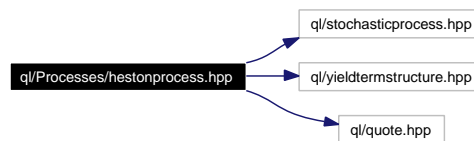
Heston stochastic process.

```
#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.

8.307 ql/Processes/merton76process.hpp File Reference

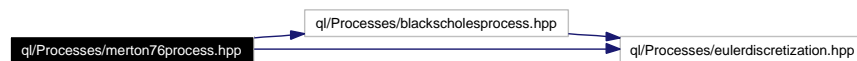
8.307.1 Detailed Description

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.

8.308 ql/Processes/ornsteinuhlenbeckprocess.hpp File Reference

8.308.1 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.

8.309 ql/Processes/squarerootprocess.hpp File Reference

8.309.1 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.

8.310 ql/Processes/stochasticprocessarray.hpp File Reference

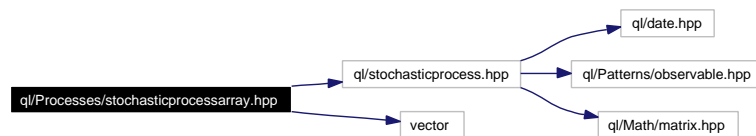
8.310.1 Detailed Description

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.

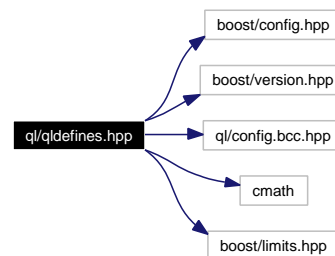
8.311 ql/qldefines.hpp File Reference

8.311.1 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** "0.3.11"
version string
- #define **QL_HEX_VERSION** 0x000311f0
version hexadecimal number
- #define **QL_LIB_VERSION** "0_3_11"
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](#)

8.312 ql/quote.hpp File Reference

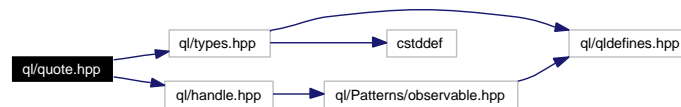
8.312.1 Detailed Description

purely virtual base class for market observables

```
#include <ql/types.hpp>
```

```
#include <ql/handle.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

8.313 ql/RandomNumbers/boxmullergaussianrng.hpp File Reference

8.313.1 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.

8.314 ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference

8.314.1 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.

8.315 ql/RandomNumbers/faurersg.hpp File Reference

8.315.1 Detailed Description

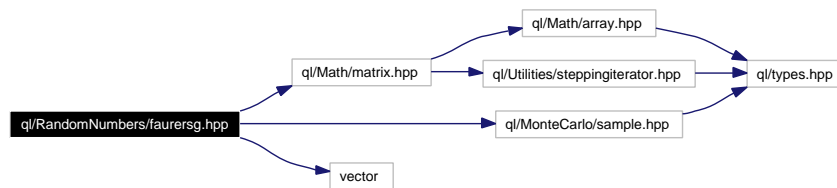
Faure low-discrepancy sequence generator.

```
#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.

8.316 ql/RandomNumbers/haltonrsg.hpp File Reference

8.316.1 Detailed Description

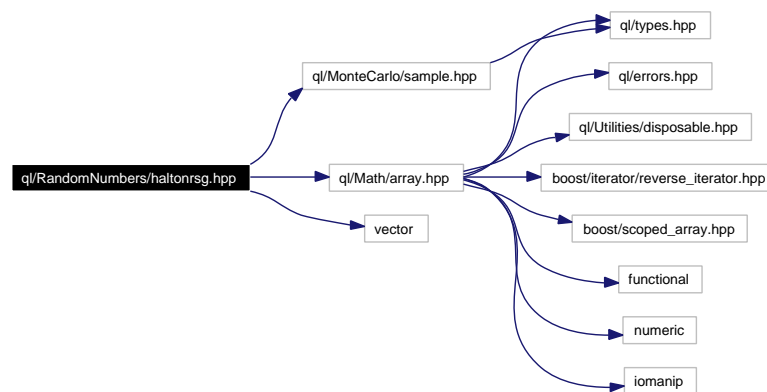
Halton low-discrepancy sequence generator.

```
#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.

8.317 ql/RandomNumbers/inversecumulativerng.hpp File Reference

8.317.1 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.

8.318 ql/RandomNumbers/inversecumulativergs.hpp File Reference

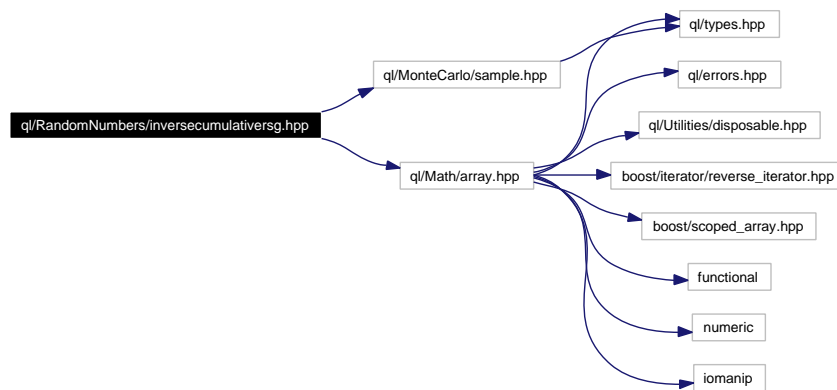
8.318.1 Detailed Description

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.

8.319 ql/RandomNumbers/knuthuniformrng.hpp File Reference

8.319.1 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.

8.320 ql/RandomNumbers/lecuyeruniformrng.hpp File Reference

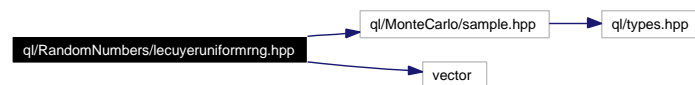
8.320.1 Detailed Description

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.

8.321 ql/RandomNumbers/mt19937uniformrng.hpp File Reference

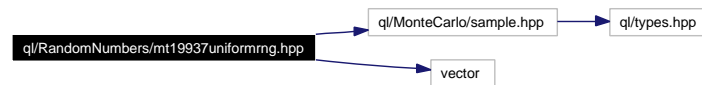
8.321.1 Detailed Description

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.

8.322 ql/RandomNumbers/randomizedlds.hpp File Reference

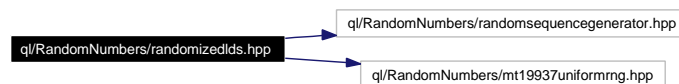
8.322.1 Detailed Description

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.

8.323 ql/RandomNumbers/randomsequencegenerator.hpp File Reference

8.323.1 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.

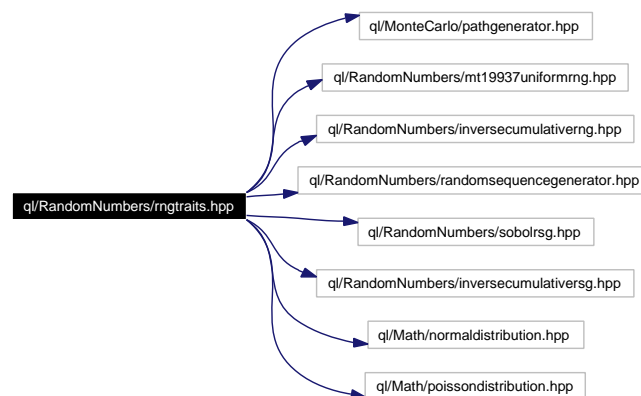
8.324 ql/RandomNumbers/rngtraits.hpp File Reference

8.324.1 Detailed Description

random-number generation policies

```
#include <ql/MonteCarlo/pathgenerator.hpp>
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
#include <ql/RandomNumbers/inversecumulativrng.hpp>
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
#include <ql/RandomNumbers/sobolrsg.hpp>
#include <ql/RandomNumbers/inversecumulativsg.hpp>
#include <ql/Math/normaldistribution.hpp>
#include <ql/Math/poissondistribution.hpp>
```

Include dependency graph for rngtraits.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef GenericPseudoRandom< MersenneTwisterUniformRng, InverseCumulativeNormal > [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

8.325 ql/RandomNumbers/seedgenerator.hpp File Reference

8.325.1 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.

8.326 ql/RandomNumbers/sobolrsg.hpp File Reference

8.326.1 Detailed Description

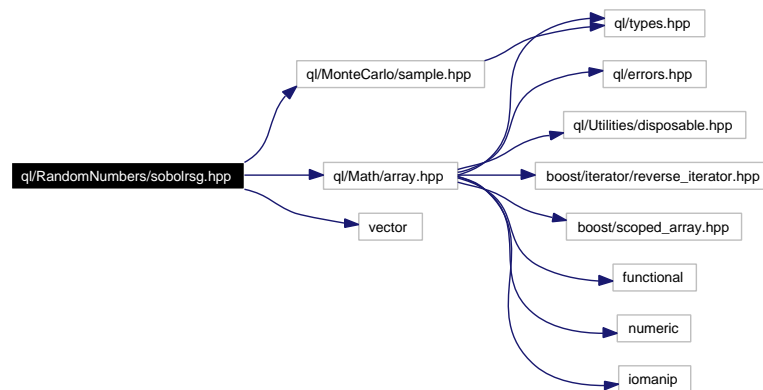
Sobol low-discrepancy sequence generator.

```
#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.

8.327 ql/schedule.hpp File Reference

8.327.1 Detailed Description

date schedule

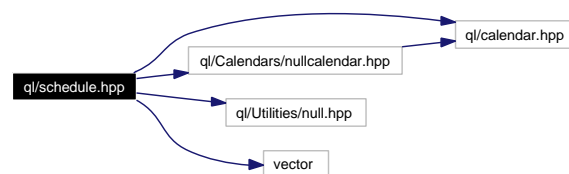
```
#include <ql/calendar.hpp>
```

```
#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

8.328 ql/settings.hpp File Reference

8.328.1 Detailed Description

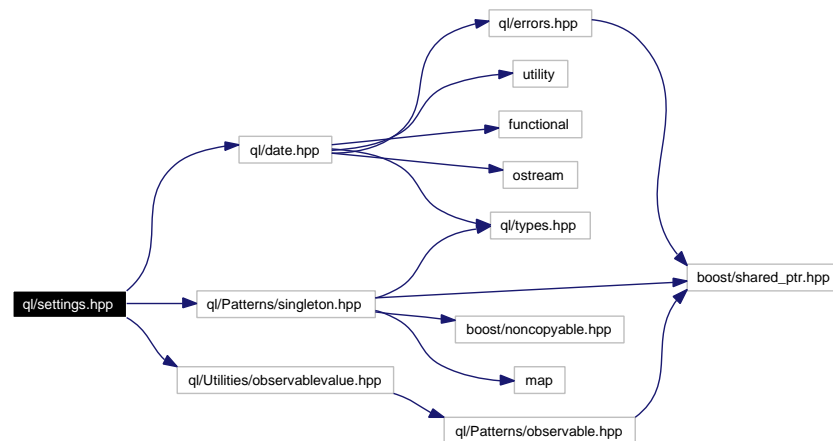
global repository for run-time library settings

```
#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)`

8.329 ql/ShortRateModels/calibrationhelper.hpp File Reference

8.329.1 Detailed Description

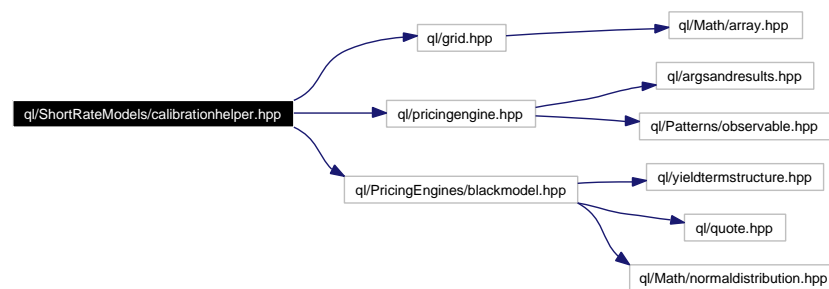
Calibration helper class.

```
#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

8.330 ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference

8.330.1 Detailed Description

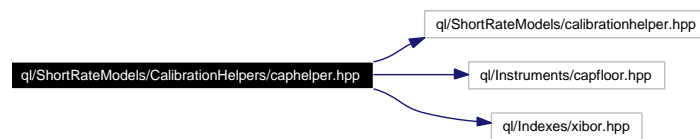
CapHelper calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for caphelper.hpp:



Namespaces

- namespace **QuantLib**

8.331 ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp File Reference

8.331.1 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

8.332 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference

8.332.1 Detailed Description

Swaption calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for swaptionhelper.hpp:



Namespaces

- namespace **QuantLib**

8.333 ql/ShortRateModels/model.hpp File Reference

8.333.1 Detailed Description

Abstract interest rate model class.

```
#include <ql/option.hpp>
```

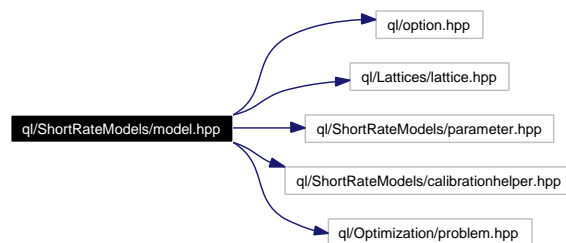
```
#include <ql/Lattices/lattice.hpp>
```

```
#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.

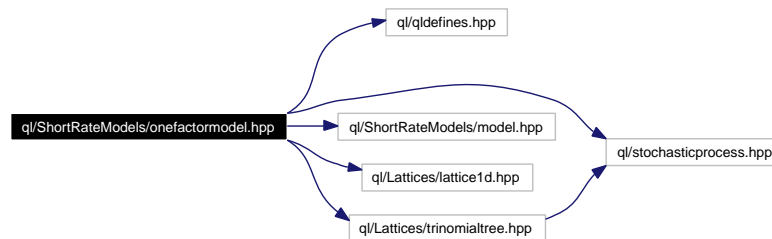
8.334 ql/ShortRateModels/onefactormodel.hpp File Reference

8.334.1 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.

8.335 ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference

8.335.1 Detailed Description

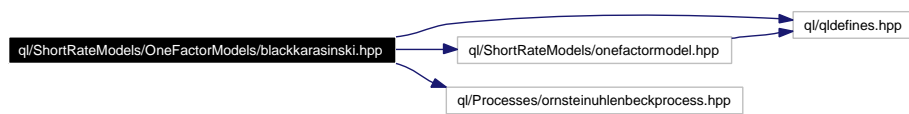
Black-Karasinski model.

```
#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.

8.336 ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference

8.336.1 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

8.337 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference

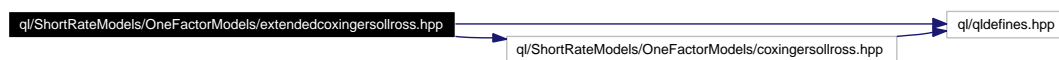
8.337.1 Detailed Description

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 $\varphi(t)$.

8.338 ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference

8.338.1 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 $\varphi(t)$.

8.339 ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference

8.339.1 Detailed Description

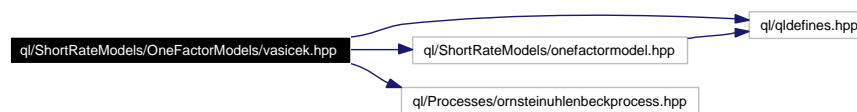
Vasicek model class.

```
#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.

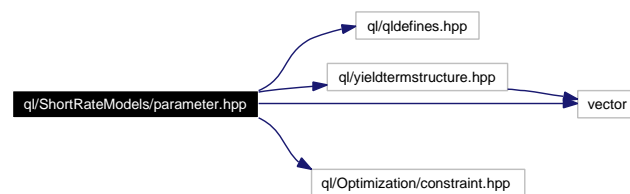
8.340 ql/ShortRateModels/parameter.hpp File Reference

8.340.1 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.

8.341 ql/ShortRateModels/twofactormodel.hpp File Reference

8.341.1 Detailed Description

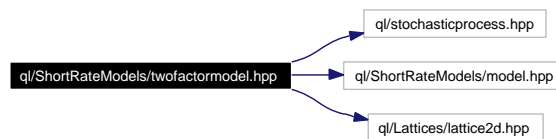
Abstract two-factor interest rate model class.

```
#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.

8.342 ql/ShortRateModels/TwoFactorModels/batesmodel.hpp File Reference

8.342.1 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](#)

8.343 ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference

8.343.1 Detailed Description

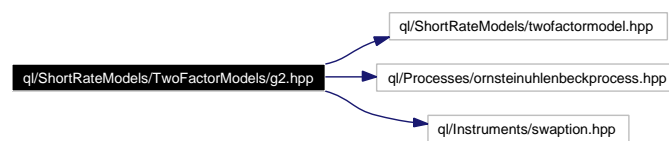
Two-factor additive Gaussian Model G2++.

```
#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 $\varphi(t)$.

8.344 ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp File Reference

8.344.1 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.

8.345 ql/solver1d.hpp File Reference

8.345.1 Detailed Description

Abstract 1-D solver class.

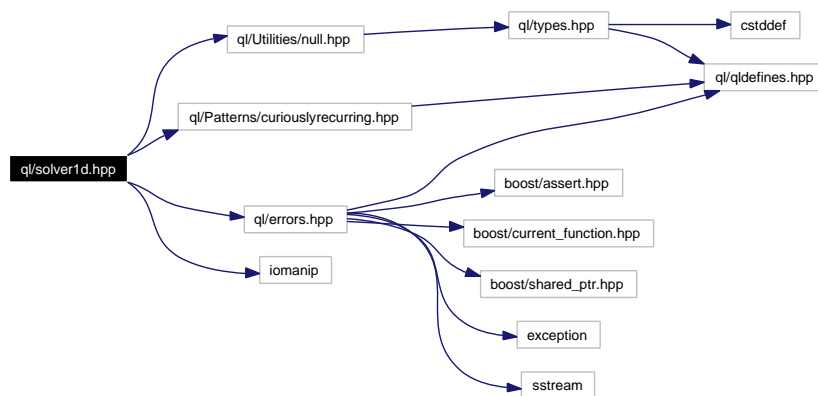
```
#include <ql/Utilities/null.hpp>
```

```
#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

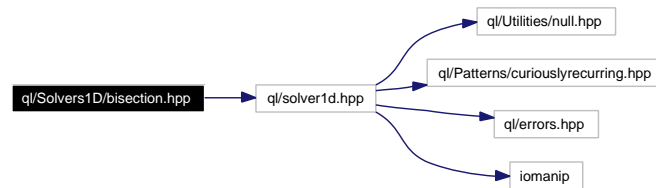
8.346 ql/Solvers1D/bisection.hpp File Reference

8.346.1 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

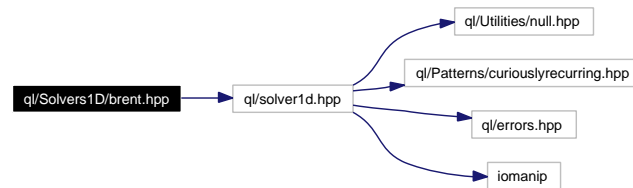
8.347 ql/Solvers1D/brent.hpp File Reference

8.347.1 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

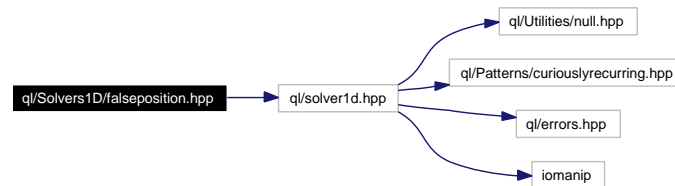
8.348 ql/Solvers1D/falseposition.hpp File Reference

8.348.1 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.

8.349 ql/Solvers1D/newton.hpp File Reference

8.349.1 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

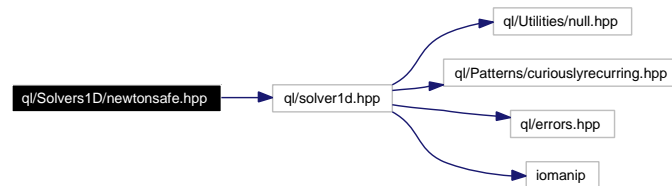
8.350 ql/Solvers1D/newtonsafe.hpp File Reference

8.350.1 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

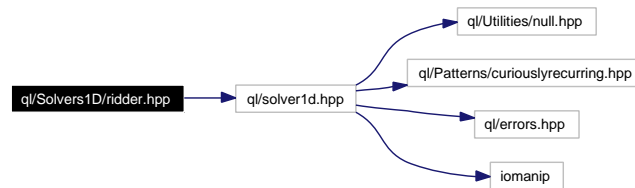
8.351 ql/Solvers1D/ridder.hpp File Reference

8.351.1 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

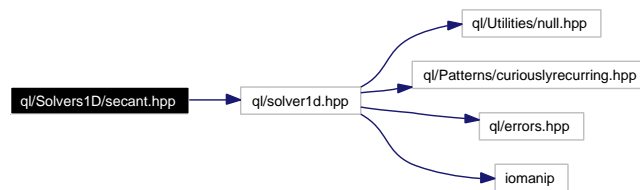
8.352 ql/Solvers1D/secant.hpp File Reference

8.352.1 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

8.353 ql/stochasticprocess.hpp File Reference

8.353.1 Detailed Description

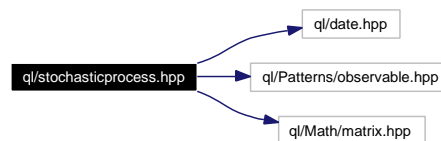
stochastic processes

```
#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

Typedefs

- typedef `StochasticProcess` [QuantLib::GenericStochasticProcess](#)

8.354 ql/swaptionvolstructure.hpp File Reference

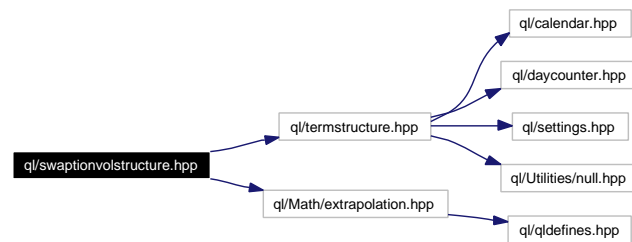
8.354.1 Detailed Description

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

8.355 ql/termstructure.hpp File Reference

8.355.1 Detailed Description

base class for term structures

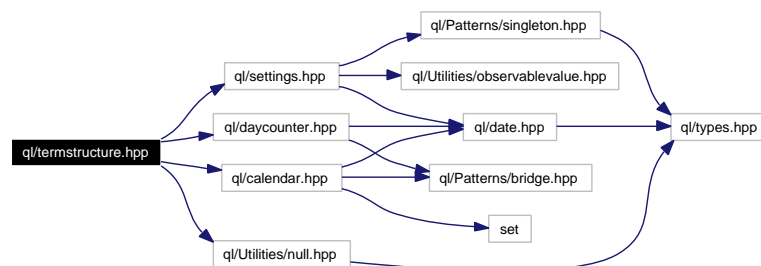
```
#include <ql/calendar.hpp>
```

```
#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.

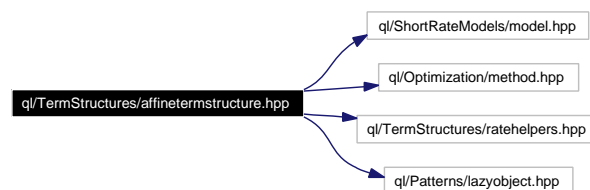
8.356 ql/TermStructures/affinetermstructure.hpp File Reference

8.356.1 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.

8.357 ql/TermStructures/bondhelpers.hpp File Reference

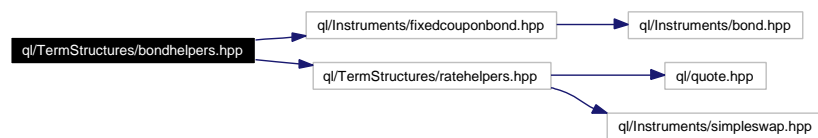
8.357.1 Detailed Description

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

8.358 ql/TermStructures/bootstraptraits.hpp File Reference

8.358.1 Detailed Description

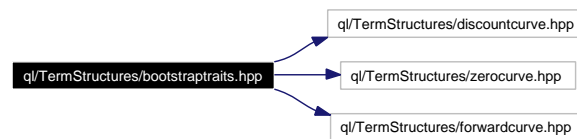
bootstrap traits

```
#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.

8.359 ql/TermStructures/compoundforward.hpp File Reference

8.359.1 Detailed Description

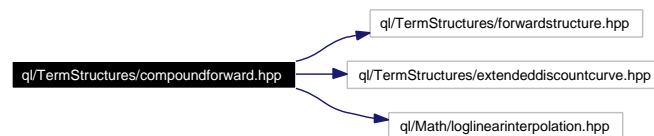
compounded forward term structure

```
#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

8.360 ql/TermStructures/discountcurve.hpp File Reference

8.360.1 Detailed Description

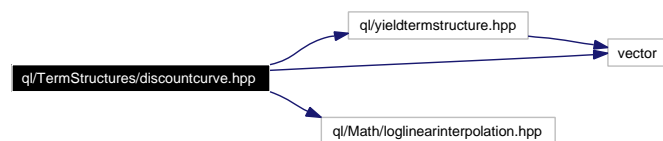
interpolated discount factor structure

```
#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.

8.361 ql/TermStructures/drifttermstructure.hpp File Reference

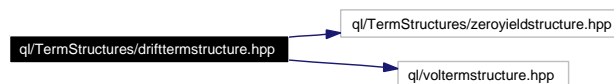
8.361.1 Detailed Description

Drift term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for drifttermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DriftTermStructure](#)
Drift term structure.

8.362 ql/TermStructures/extendeddiscountcurve.hpp File Reference

8.362.1 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.

8.363 ql/TermStructures/flatforward.hpp File Reference

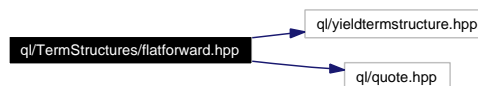
8.363.1 Detailed Description

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.

8.364 ql/TermStructures/forwardcurve.hpp File Reference

8.364.1 Detailed Description

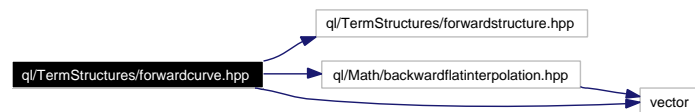
interpolated forward-rate structure

```
#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.

8.365 ql/TermStructures/forwardspreadedtermstructure.hpp File Reference

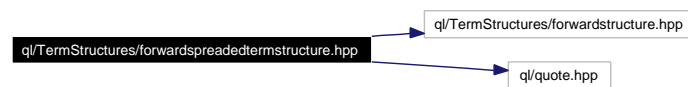
8.365.1 Detailed Description

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.

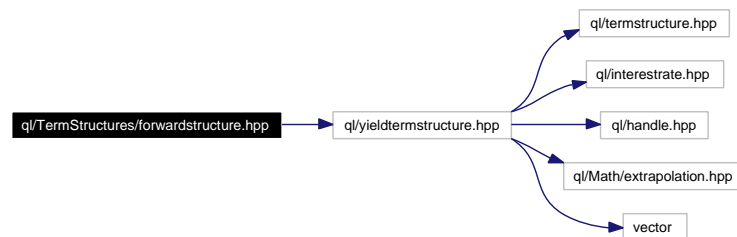
8.366 ql/TermStructures/forwardstructure.hpp File Reference

8.366.1 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.

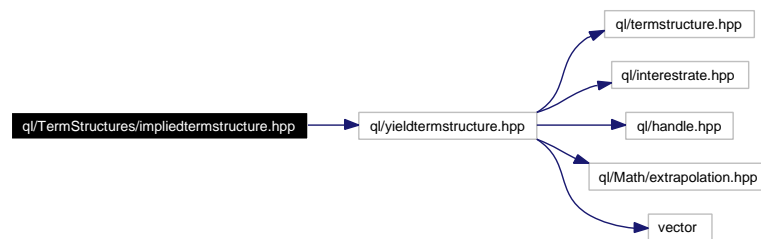
8.367 ql/TermStructures/IMPLIEDTERMSTRUCTURE.hpp File Reference

8.367.1 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.

8.368 ql/TermStructures/piecewiseflatforward.hpp File Reference

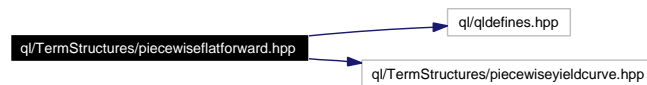
8.368.1 Detailed Description

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.

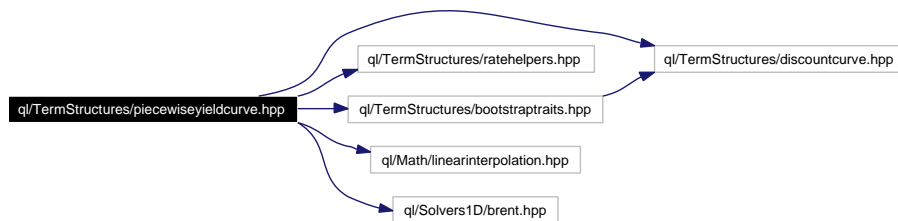
8.369 ql/TermStructures/piecewiseyieldcurve.hpp File Reference

8.369.1 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.

8.370 ql/TermStructures/quantotermstructure.hpp File Reference

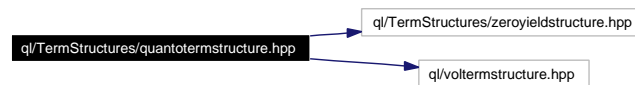
8.370.1 Detailed Description

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.

8.371 ql/TermStructures/ratehelpers.hpp File Reference

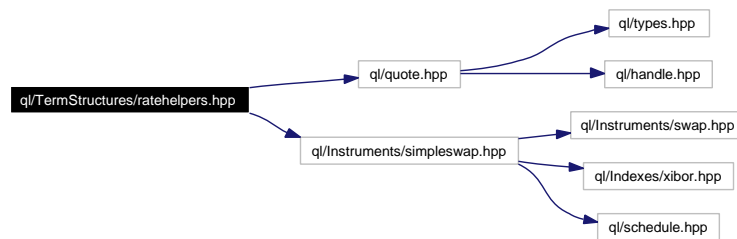
8.371.1 Detailed Description

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.
- class [FraRateHelper](#)
Forward rate agreement.
- class [FuturesRateHelper](#)
Interest-rate futures.
- class [SwapRateHelper](#)
Swap rate

8.372 ql/TermStructures/zerocurve.hpp File Reference

8.372.1 Detailed Description

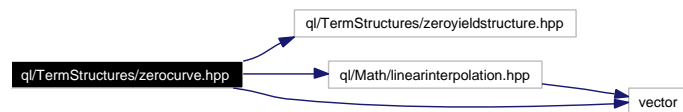
interpolated zero-rates structure

```
#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.

8.373 ql/TermStructures/zerospreadedtermstructure.hpp File Reference

8.373.1 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.

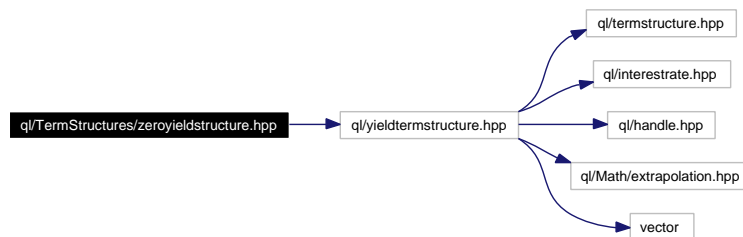
8.374 ql/TermStructures/zeroyieldstructure.hpp File Reference

8.374.1 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.

8.375 ql/timegrid.hpp File Reference

8.375.1 Detailed Description

discrete time grid

```
#include <ql/types.hpp>
```

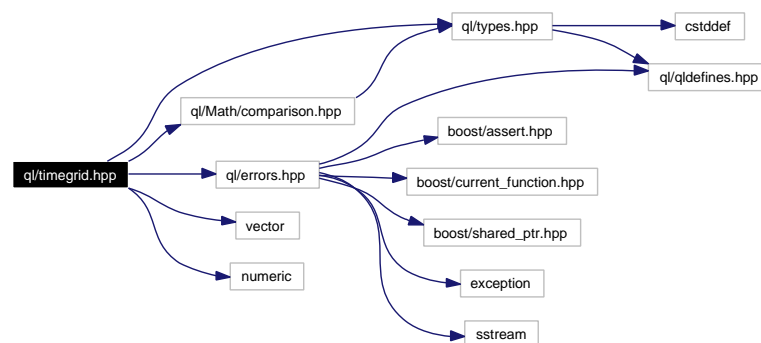
```
#include <ql/errors.hpp>
```

```
#include <ql/Math/comparison.hpp>
```

```
#include <vector>
```

```
#include <numeric>
```

Include dependency graph for timegrid.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TimeGrid](#)
time grid class

8.376 ql/types.hpp File Reference

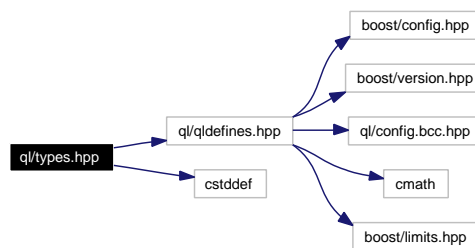
8.376.1 Detailed Description

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

8.377 ql/Utilities/dataformatters.hpp File Reference

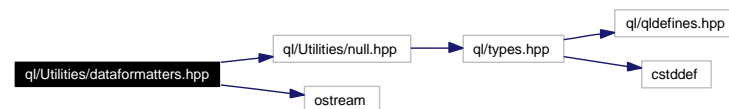
8.377.1 Detailed Description

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

8.378 ql/Utilities/dataparsers.hpp File Reference

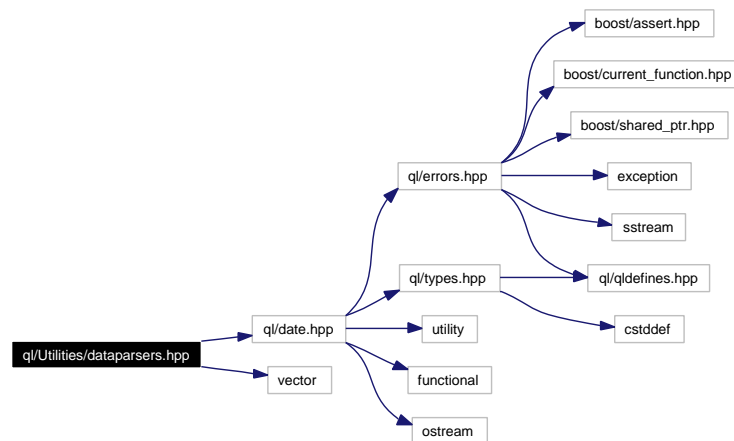
8.378.1 Detailed Description

Classes used to parse data for input.

```
#include <ql/date.hpp>
```

```
#include <vector>
```

Include dependency graph for dataparsers.hpp:



Namespaces

- namespace **QuantLib**

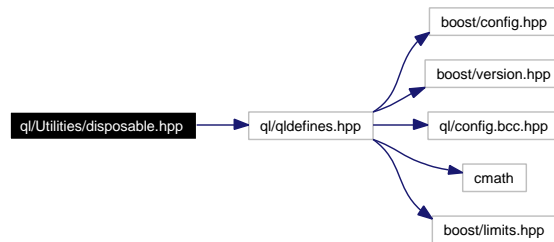
8.379 ql/Utilities/disposable.hpp File Reference

8.379.1 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

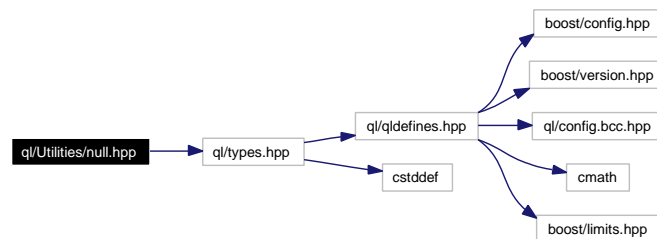
8.380 ql/Utilities/null.hpp File Reference

8.380.1 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.

8.381 ql/Utilities/observablevalue.hpp File Reference

8.381.1 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

8.382 ql/Utilities/steppingiterator.hpp File Reference

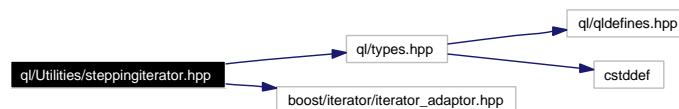
8.382.1 Detailed Description

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.

8.383 ql/Utilities/strings.hpp File Reference

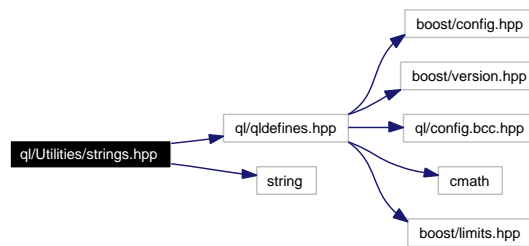
8.383.1 Detailed Description

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 &`)

8.384 ql/Utilities/tracing.hpp File Reference

8.384.1 Detailed Description

tracing facilities

```
#include <ql/types.hpp>
```

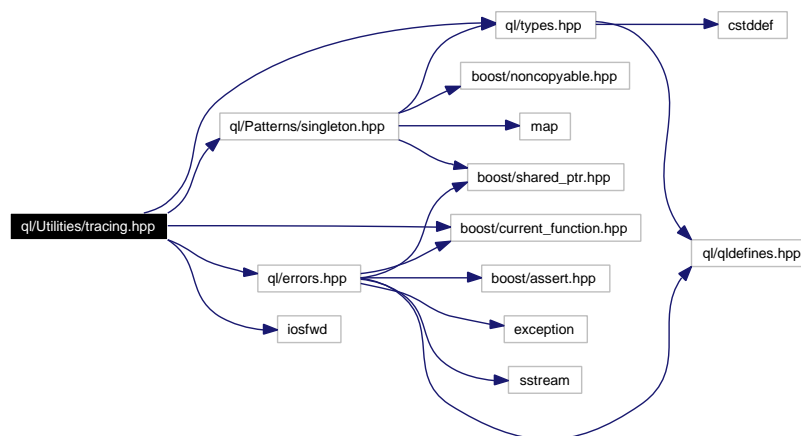
```
#include <ql/errors.hpp>
```

```
#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

8.385 ql/Volatilities/blackconstantvol.hpp File Reference

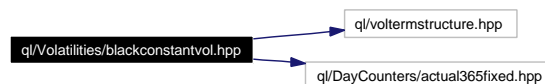
8.385.1 Detailed Description

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.

8.386 ql/Volatilities/blackvariancecurve.hpp File Reference

8.386.1 Detailed Description

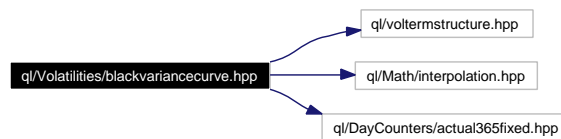
Black volatility curve modelled as variance curve.

```
#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.

8.387 ql/Volatilities/blackvariancesurface.hpp File Reference

8.387.1 Detailed Description

Black volatility surface modelled as variance surface.

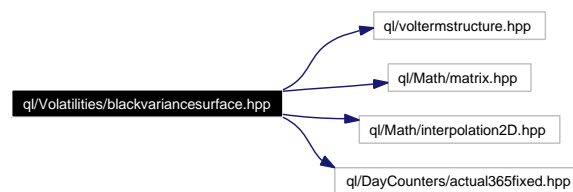
```
#include <ql/voltermstructure.hpp>
```

```
#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.

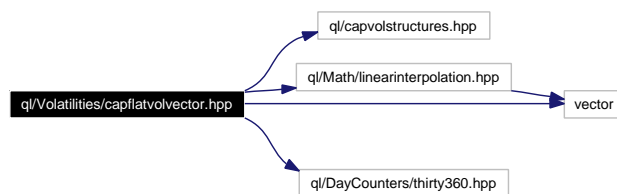
8.388 ql/Volatilities/capflatvolvector.hpp File Reference

8.388.1 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.

8.389 ql/Volatilities/capletconstantvol.hpp File Reference

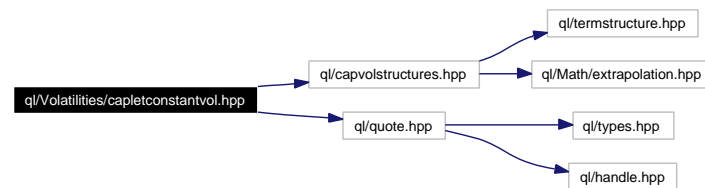
8.389.1 Detailed Description

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.

8.390 ql/Volatilities/capletvariancecurve.hpp File Reference

8.390.1 Detailed Description

caplet variance curve

```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for capletvariancecurve.hpp:



Namespaces

- namespace **QuantLib**

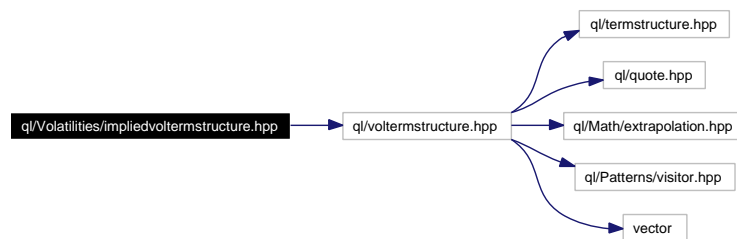
8.391 ql/Volatilities/impliedvoltermstructure.hpp File Reference

8.391.1 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.

8.392 ql/Volatilities/localconstantvol.hpp File Reference

8.392.1 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.

8.393 ql/Volatilities/localvolcurve.hpp File Reference

8.393.1 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.

8.394 ql/Volatilities/localvolsurface.hpp File Reference

8.394.1 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.

8.395 ql/Volatilities/swaptionvolmatrix.hpp File Reference

8.395.1 Detailed Description

Swaption at-the-money volatility matrix.

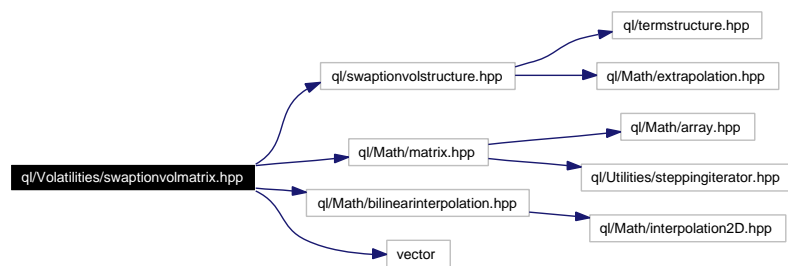
```
#include <ql/swaptionvolstructure.hpp>
```

```
#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.

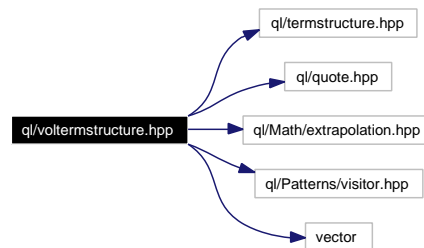
8.396 ql/voltermstructure.hpp File Reference

8.396.1 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.

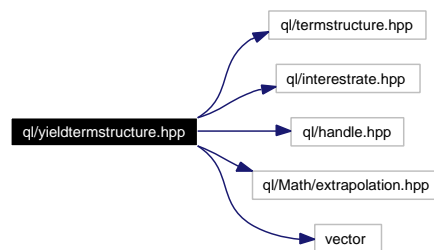
8.397 ql/yieldtermstructure.hpp File Reference

8.397.1 Detailed Description

Interest-rate term structure.

```
#include <ql/termstructure.hpp>
#include <ql/interestrates.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 AmericanOption.cpp

This example calculates American options using different methods

```
1 /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
22 #include <ql/quantlib.hpp>
23 #include <iostream>
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         std::cout << "Using " << QL_VERSION << std::endl << std::endl;
42
43         // our option
44         Option::Type type(Option::Put);
45         Real underlying = 36;
46         Real strike = 40;
47         Spread dividendYield = 0.00;
48         Rate riskFreeRate = 0.06;
49         Volatility volatility = 0.20;
50
51         Date todaysDate(15, May, 1998);
52         Date settlementDate(17, May, 1998);
53         Settings::instance().evaluationDate() = todaysDate;
54
55         Date exerciseDate(17, May, 1999);
56         DayCounter dayCounter = Actual365Fixed();
57         Time maturity = dayCounter.yearFraction(settlementDate,
58                                                 exerciseDate);
59
60         std::cout << "option type = " << type << std::endl;
```

```

61     std::cout << "Time to maturity = " << maturity
62         << std::endl;
63     std::cout << "Underlying price = " << underlying
64         << std::endl;
65     std::cout << "Strike = " << strike
66         << std::endl;
67     std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
68         << std::endl;
69     std::cout << "Dividend yield = " << io::rate(dividendYield)
70         << std::endl;
71     std::cout << "Volatility = " << io::volatility(volatility)
72         << std::endl;
73     std::cout << std::endl;
74
75     std::string method;
76
77     Real value, discrepancy, rightValue, relativeDiscrepancy;
78     rightValue = (type == Option::Put ? 4.48667344 : 2.17372645);
79
80     std::cout << std::endl ;
81
82     // write column headings
83     Size widths[] = { 35, 14, 14, 14 };
84     std::cout << std::setw(widths[0]) << std::left << "Method"
85         << std::setw(widths[1]) << std::left << "Value"
86         << std::setw(widths[2]) << std::left << "Discrepancy"
87         << std::setw(widths[3]) << std::left << "Rel. Discr."
88         << std::endl;
89
90     Date midlifeDate(19, November, 1998);
91     std::vector<Date> exDates(2);
92     exDates[0]=midlifeDate;
93     exDates[1]=exerciseDate;
94
95     boost::shared_ptr<Exercise> exercise(
96         new EuropeanExercise(exerciseDate));
97     boost::shared_ptr<Exercise> amExercise(
98         new AmericanExercise(settlementDate,
99             exerciseDate));
100     boost::shared_ptr<Exercise> berExercise(new BermudanExercise(exDates));
101
102
103     Handle<Quote> underlyingH(
104         boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
105
106     // bootstrap the yield/dividend/vol curves
107     Handle<YieldTermStructure> flatTermStructure(
108         boost::shared_ptr<YieldTermStructure>(
109             new FlatForward(settlementDate, riskFreeRate, dayCounter)));
110     Handle<YieldTermStructure> flatDividendTS(
111         boost::shared_ptr<YieldTermStructure>(
112             new FlatForward(settlementDate, dividendYield, dayCounter)));
113     Handle<BlackVolTermStructure> flatVolTS(
114         boost::shared_ptr<BlackVolTermStructure>(
115             new BlackConstantVol(settlementDate, volatility, dayCounter)));
116
117     std::vector<Date> dates(4);
118     dates[0] = settlementDate + 1*Months;
119     dates[1] = exerciseDate;
120     dates[2] = exerciseDate + 6*Months;
121     dates[3] = exerciseDate + 12*Months;
122     std::vector<Real> strikes(4);
123     strikes[0] = underlying*0.9;
124     strikes[1] = underlying;
125     strikes[2] = underlying*1.1;
126     strikes[3] = underlying*1.2;
127

```

```

128     Matrix vols(4,4);
129     vols[0][0] = volatility*1.1; vols[0][1] = volatility;
130     vols[0][2] = volatility*0.9; vols[0][3] = volatility*0.8;
131     vols[1][0] = volatility*1.1; vols[1][1] = volatility;
132     vols[1][2] = volatility*0.9; vols[1][3] = volatility*0.8;
133     vols[2][0] = volatility*1.1; vols[2][1] = volatility;
134     vols[2][2] = volatility*0.9; vols[2][3] = volatility*0.8;
135     vols[3][0] = volatility*1.1; vols[3][1] = volatility;
136     vols[3][2] = volatility*0.9; vols[3][3] = volatility*0.8;
137     Handle<BlackVolTermStructure> blackSurface(
138         boost::shared_ptr<BlackVolTermStructure>(new
139             BlackVarianceSurface(settlementDate, dates,
140                 strikes, vols, dayCounter)));
141
142     boost::shared_ptr<StrikedTypePayoff> payoff(new
143         PlainVanillaPayoff(type, strike));
144
145     boost::shared_ptr<BlackScholesProcess> stochasticProcess(new
146         BlackScholesProcess(
147             underlyingH,
148             flatDividendTS,
149             flatTermStructure,
150             flatVolTS));
151
152     // European option
153     VanillaOption euroOption(stochasticProcess, payoff, exercise,
154         boost::shared_ptr<PricingEngine>(new AnalyticEuropeanEngine()));
155
156     // method: Black Scholes Engine
157     method = "equivalent European option";
158     value = euroOption.NPV();
159     std::cout << std::setw(widths[0]) << std::left << method
160         << std::fixed
161         << std::setw(widths[1]) << std::left << value
162         << std::setw(widths[2]) << std::left << "N/A"
163         << std::setw(widths[3]) << std::left << "N/A"
164         << std::endl;
165
166     // American option
167     VanillaOption option(stochasticProcess, payoff, amExercise);
168
169     // target value
170     method = "reference value";
171     std::cout << std::setw(widths[0]) << std::left << method
172         << std::fixed
173         << std::setw(widths[1]) << std::left << rightValue
174         << std::setw(widths[2]) << std::left << "N/A"
175         << std::setw(widths[3]) << std::left << "N/A"
176         << std::endl;
177
178     Size timeSteps = 801;
179
180     // Finite differences
181     method = "Finite differences";
182     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
183         new FDAmericanEngine(timeSteps, timeSteps-1)));
184     value = option.NPV();
185     discrepancy = std::fabs(value-rightValue);
186     relativeDiscrepancy = discrepancy/rightValue;
187     std::cout << std::setw(widths[0]) << std::left << method
188         << std::fixed
189         << std::setw(widths[1]) << std::left << value
190         << std::setw(widths[2]) << std::left << discrepancy
191         << std::scientific
192         << std::setw(widths[3]) << std::left << relativeDiscrepancy
193         << std::endl;
194

```

```

195 // Binomial Method (JR)
196 method = "Binomial Jarrow-Rudd";
197 option.setPricingEngine(boost::shared_ptr<PricingEngine>(
198     new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
199 value = option.NPV();
200 discrepancy = std::fabs(value-rightValue);
201 relativeDiscrepancy = discrepancy/rightValue;
202 std::cout << std::setw(widths[0]) << std::left << method
203     << std::fixed
204     << std::setw(widths[1]) << std::left << value
205     << std::setw(widths[2]) << std::left << discrepancy
206     << std::scientific
207     << std::setw(widths[3]) << std::left << relativeDiscrepancy
208     << std::endl;
209
210 // Binomial Method (CRR)
211 method = "Binomial Cox-Ross-Rubinstein";
212 option.setPricingEngine(boost::shared_ptr<PricingEngine>(
213     new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
214 value = option.NPV();
215 discrepancy = std::fabs(value-rightValue);
216 relativeDiscrepancy = discrepancy/rightValue;
217 std::cout << std::setw(widths[0]) << std::left << method
218     << std::fixed
219     << std::setw(widths[1]) << std::left << value
220     << std::setw(widths[2]) << std::left << discrepancy
221     << std::scientific
222     << std::setw(widths[3]) << std::left << relativeDiscrepancy
223     << std::endl;
224
225 // Equal Probability Additive Binomial Tree (EQP)
226 method = "Additive equiprobabilities";
227 option.setPricingEngine(boost::shared_ptr<PricingEngine>(
228     new BinomialVanillaEngine<AdditiveEQPBinoomialTree>(timeSteps)));
229 value = option.NPV();
230 discrepancy = std::fabs(value-rightValue);
231 relativeDiscrepancy = discrepancy/rightValue;
232 std::cout << std::setw(widths[0]) << std::left << method
233     << std::fixed
234     << std::setw(widths[1]) << std::left << value
235     << std::setw(widths[2]) << std::left << discrepancy
236     << std::scientific
237     << std::setw(widths[3]) << std::left << relativeDiscrepancy
238     << std::endl;
239
240 // Equal Jumps Additive Binomial Tree (Trigeorgis)
241 method = "Binomial Trigeorgis";
242 option.setPricingEngine(boost::shared_ptr<PricingEngine>(
243     new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
244 value = option.NPV();
245 discrepancy = std::fabs(value-rightValue);
246 relativeDiscrepancy = discrepancy/rightValue;
247 std::cout << std::setw(widths[0]) << std::left << method
248     << std::fixed
249     << std::setw(widths[1]) << std::left << value
250     << std::setw(widths[2]) << std::left << discrepancy
251     << std::scientific
252     << std::setw(widths[3]) << std::left << relativeDiscrepancy
253     << std::endl;
254
255 // Tian Binomial Tree (third moment matching)
256 method = "Binomial Tian";
257 option.setPricingEngine(boost::shared_ptr<PricingEngine>(
258     new BinomialVanillaEngine<Tian>(timeSteps)));
259 value = option.NPV();
260 discrepancy = std::fabs(value-rightValue);
261 relativeDiscrepancy = discrepancy/rightValue;

```

```

262         std::cout << std::setw(widths[0]) << std::left << method
263             << std::fixed
264             << std::setw(widths[1]) << std::left << value
265             << std::setw(widths[2]) << std::left << discrepancy
266             << std::scientific
267             << std::setw(widths[3]) << std::left << relativeDiscrepancy
268             << std::endl;
269
270         // Leisen-Reimer Binomial Tree
271         method = "Binomial Leisen-Reimer";
272         option.setPricingEngine(boost::shared_ptr<PricingEngine>(
273             new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
274         value = option.NPV();
275         discrepancy = std::fabs(value-rightValue);
276         relativeDiscrepancy = discrepancy/rightValue;
277         std::cout << std::setw(widths[0]) << std::left << method
278             << std::fixed
279             << std::setw(widths[1]) << std::left << value
280             << std::setw(widths[2]) << std::left << discrepancy
281             << std::scientific
282             << std::setw(widths[3]) << std::left << relativeDiscrepancy
283             << std::endl;
284
285         // Barone-Adesi and Whaley approximation
286         method = "Barone-Adesi and Whaley approx.";
287         option.setPricingEngine(boost::shared_ptr<PricingEngine>(
288             new BaroneAdesiWhaleyApproximationEngine));
289         value = option.NPV();
290         discrepancy = std::fabs(value-rightValue);
291         relativeDiscrepancy = discrepancy/rightValue;
292         std::cout << std::setw(widths[0]) << std::left << method
293             << std::fixed
294             << std::setw(widths[1]) << std::left << value
295             << std::setw(widths[2]) << std::left << discrepancy
296             << std::scientific
297             << std::setw(widths[3]) << std::left << relativeDiscrepancy
298             << std::endl;
299
300         // Bjerksund and Stensland approximation
301         method = "Bjerksund and Stensland approx.";
302         option.setPricingEngine(boost::shared_ptr<PricingEngine>(
303             new BjerksundStenslandApproximationEngine));
304         value = option.NPV();
305         discrepancy = std::fabs(value-rightValue);
306         relativeDiscrepancy = discrepancy/rightValue;
307         std::cout << std::setw(widths[0]) << std::left << method
308             << std::fixed
309             << std::setw(widths[1]) << std::left << value
310             << std::setw(widths[2]) << std::left << discrepancy
311             << std::scientific
312             << std::setw(widths[3]) << std::left << relativeDiscrepancy
313             << std::endl;
314
315         return 0;
316     } catch (std::exception& e) {
317         std::cout << e.what() << std::endl;
318         return 1;
319     } catch (...) {
320         std::cout << "unknown error" << std::endl;
321         return 1;
322     }
323 }

```

9.2 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
21 #include <ql/quantlib.hpp>
22 #include <iostream>
23
24 using namespace QuantLib;
25
26 #if defined(QL_ENABLE_SESSIONS)
27 namespace QuantLib {
28
29     Integer sessionId() { return 0; }
30
31 }
32 #endif
33
34
35 //Number of swaptions to be calibrated to...
36
37 Size numRows = 5;
38 Size numCols = 5;
39
40 Integer swapLengths[] = {
41     1,    2,    3,    4,    5};
42 Volatility swaptionVols[] = {
43     0.1490, 0.1340, 0.1228, 0.1189, 0.1148,
44     0.1290, 0.1201, 0.1146, 0.1108, 0.1040,
45     0.1149, 0.1112, 0.1070, 0.1010, 0.0957,
46     0.1047, 0.1021, 0.0980, 0.0951, 0.1270,
47     0.1000, 0.0950, 0.0900, 0.1230, 0.1160};
48
49 void calibrateModel(const boost::shared_ptr<ShortRateModel>& model,
50                     const std::vector<boost::shared_ptr<CalibrationHelper> >&
51                         helpers,
52                     Real lambda) {
53
54     Simplex om(lambda, 1e-9);
55     om.setEndCriteria(EndCriteria(10000, 1e-7));
56     model->calibrate(helpers, om);
57
58     // Output the implied Black volatilities
59     for (Size i=0; i<numRows; i++) {
60         Size j = numCols - i - 1; // 1x5, 2x4, 3x3, 4x2, 5x1
61         Size k = i*numCols + j;
62         Real npv = helpers[i]->modelValue();
63         Volatility implied = helpers[i]->impliedVolatility(npv, 1e-4,
64                                                             1000, 0.05, 0.50);
65         Volatility diff = implied - swaptionVols[k];
66
67         std::cout << i+1 << "x" << swapLengths[j]
68                 << std::setprecision(5) << std::noshowpos
69                 << ": model " << std::setw(7) << io::volatility(implied)
70                 << ", market " << std::setw(7)
71                 << io::volatility(swaptionVols[k])
72                 << " (" << std::setw(7) << std::showpos
73                 << io::volatility(diff) << std::noshowpos << ")\n";
74     }
75 }
76
77 int main(int, char* [])
78 {
79     try {
80         QL_IO_INIT

```

```

81
82     Date todaysDate(15, February, 2002);
83     Calendar calendar = TARGET();
84     Date settlementDate(19, February, 2002);
85     Settings::instance().evaluationDate() = todaysDate;
86
87     // flat yield term structure impling 1x5 swap at 5%
88     boost::shared_ptr<Quote> flatRate(new SimpleQuote(0.04875825));
89     boost::shared_ptr<FlatForward> myTermStructure(
90         new FlatForward(settlementDate, Handle<Quote>(flatRate),
91             Actual365Fixed()));
92     Handle<YieldTermStructure> rhTermStructure;
93     rhTermStructure.linkTo(myTermStructure);
94
95     // Define the ATM/OTM/ITM swaps
96     Frequency fixedLegFrequency = Annual;
97     BusinessDayConvention fixedLegConvention = Unadjusted;
98     BusinessDayConvention floatingLegConvention = ModifiedFollowing;
99     DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
100    Frequency floatingLegFrequency = Semiannual;
101    bool payFixedRate = true;
102    Integer fixingDays = 2;
103    Rate dummyFixedRate = 0.03;
104    boost::shared_ptr<Xibor> indexSixMonths(new
105        Euribor(6, Months, rhTermStructure));
106
107    Date startDate = calendar.advance(settlementDate,1,Years,
108        floatingLegConvention);
109    Date maturity = calendar.advance(startDate,5,Years,
110        floatingLegConvention);
111    Schedule fixedSchedule(calendar,startDate,maturity,
112        fixedLegFrequency,fixedLegConvention);
113    Schedule floatSchedule(calendar,startDate,maturity,
114        floatingLegFrequency,floatingLegConvention);
115    boost::shared_ptr<SimpleSwap> swap(new SimpleSwap(
116        payFixedRate, 1000.0,
117        fixedSchedule, dummyFixedRate, fixedLegDayCounter,
118        floatSchedule, indexSixMonths, fixingDays, 0.0,
119        rhTermStructure));
120    Rate fixedATMRate = swap->fairRate();
121    Rate fixedOTMRate = fixedATMRate * 1.2;
122    Rate fixedITMRate = fixedATMRate * 0.8;
123
124    boost::shared_ptr<SimpleSwap> atmSwap(new SimpleSwap(
125        payFixedRate, 1000.0,
126        fixedSchedule, fixedATMRate, fixedLegDayCounter,
127        floatSchedule, indexSixMonths, fixingDays, 0.0,
128        rhTermStructure));
129    boost::shared_ptr<SimpleSwap> otmSwap(new SimpleSwap(
130        payFixedRate, 1000.0,
131        fixedSchedule, fixedOTMRate, fixedLegDayCounter,
132        floatSchedule, indexSixMonths, fixingDays, 0.0,
133        rhTermStructure));
134    boost::shared_ptr<SimpleSwap> itmSwap(new SimpleSwap(
135        payFixedRate, 1000.0,
136        fixedSchedule, fixedITMRate, fixedLegDayCounter,
137        floatSchedule, indexSixMonths, fixingDays, 0.0,
138        rhTermStructure));
139
140    // defining the swaptions to be used in model calibration
141    std::vector<Period> swaptionMaturities;
142    swaptionMaturities.push_back(Period(1, Years));
143    swaptionMaturities.push_back(Period(2, Years));
144    swaptionMaturities.push_back(Period(3, Years));
145    swaptionMaturities.push_back(Period(4, Years));
146    swaptionMaturities.push_back(Period(5, Years));
147

```

```

148     std::vector<boost::shared_ptr<CalibrationHelper> > swaptions;
149
150     // List of times that have to be included in the timegrid
151     std::list<Time> times;
152
153     Size i;
154     for (i=0; i<numRows; i++) {
155         Size j = numCols - i -1; // 1x5, 2x4, 3x3, 4x2, 5x1
156         Size k = i*numCols + j;
157         boost::shared_ptr<Quote> vol(new SimpleQuote(swaptionVols[k]));
158         swaptions.push_back(boost::shared_ptr<CalibrationHelper>(new
159             SwaptionHelper(swaptionMaturities[i],
160                 Period(swapLengths[j], Years),
161                 Handle<Quote>(vol),
162                 indexSixMonths,
163                 indexSixMonths->frequency(),
164                 indexSixMonths->dayCounter(),
165                 rhTermStructure)));
166         swaptions.back()->addTimesTo(times);
167     }
168
169     // Building time-grid
170     TimeGrid grid(times.begin(), times.end(), 30);
171
172
173     // defining the models
174     boost::shared_ptr<G2> modelG2(new G2(rhTermStructure));
175     boost::shared_ptr<HullWhite> modelHW(new HullWhite(rhTermStructure));
176     boost::shared_ptr<HullWhite> modelHW2(new HullWhite(rhTermStructure));
177     boost::shared_ptr<BlackKarasinski> modelBK(
178         new BlackKarasinski(rhTermStructure));
179
180
181     // model calibrations
182
183     std::cout << "G2 (analytic formulae) calibration" << std::endl;
184     for (i=0; i<swaptions.size(); i++)
185         swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
186             new G2SwaptionEngine(modelG2, 6.0, 16)));
187
188     calibrateModel(modelG2, swaptions, 0.05);
189     std::cout << "calibrated to:\n"
190         << "a      = " << modelG2->params()[0] << ", "
191         << "sigma = " << modelG2->params()[1] << "\n"
192         << "b      = " << modelG2->params()[2] << ", "
193         << "eta    = " << modelG2->params()[3] << "\n"
194         << "rho    = " << modelG2->params()[4]
195         << std::endl << std::endl;
196
197
198
199     std::cout << "Hull-White (analytic formulae) calibration" << std::endl;
200     for (i=0; i<swaptions.size(); i++)
201         swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
202             new JamshidianSwaptionEngine(modelHW)));
203
204     calibrateModel(modelHW, swaptions, 0.05);
205     std::cout << "calibrated to:\n"
206         << "a = " << modelHW->params()[0] << ", "
207         << "sigma = " << modelHW->params()[1]
208         << std::endl << std::endl;
209
210     std::cout << "Hull-White (numerical) calibration" << std::endl;
211     for (i=0; i<swaptions.size(); i++)
212         swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
213             new TreeSwaptionEngine(modelHW2,grid)));
214

```



```

215     calibrateModel(modelHW2, swaptions, 0.05);
216     std::cout << "calibrated to:\n"
217         << "a = " << modelHW2->params()[0] << ", "
218         << "sigma = " << modelHW2->params()[1]
219         << std::endl << std::endl;
220
221     std::cout << "Black-Karasinski (numerical) calibration" << std::endl;
222     for (i=0; i<swaptions.size(); i++)
223         swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
224             new TreeSwaptionEngine(modelBK,grid)));
225
226     calibrateModel(modelBK, swaptions, 0.05);
227     std::cout << "calibrated to:\n"
228         << "a = " << modelBK->params()[0] << ", "
229         << "sigma = " << modelBK->params()[1]
230         << std::endl << std::endl;
231
232
233     // ATM Bermudan swaption pricing
234
235     std::cout << "Payer bermudan swaption "
236         << "struck at " << io::rate(fixedATMRate)
237         << " (ATM)" << std::endl;
238
239     std::vector<Date> bermudanDates;
240     const std::vector<boost::shared_ptr<CashFlow> >& leg =
241         swap->fixedLeg();
242     for (i=0; i<leg.size(); i++) {
243         boost::shared_ptr<Coupon> coupon =
244             boost::dynamic_pointer_cast<Coupon>(leg[i]);
245         bermudanDates.push_back(coupon->accrualStartDate());
246     }
247
248     boost::shared_ptr<Exercise> bermudanExercise(
249         new BermudanExercise(bermudanDates));
250
251     Swaption bermudanSwaption(atmSwap, bermudanExercise, rhTermStructure,
252         boost::shared_ptr<PricingEngine>());
253
254     // Do the pricing for each model
255
256     // G2 price the European swaption here, it should switch to bermudan
257     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
258         TreeSwaptionEngine(modelG2, 50)));
259     std::cout << "G2:      " << bermudanSwaption.NPV() << std::endl;
260
261     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
262         new TreeSwaptionEngine(modelHW, 50)));
263     std::cout << "HW:      " << bermudanSwaption.NPV() << std::endl;
264
265     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
266         TreeSwaptionEngine(modelHW2, 50)));
267     std::cout << "HW (num): " << bermudanSwaption.NPV() << std::endl;
268
269     bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
270         TreeSwaptionEngine(modelBK, 50)));
271     std::cout << "BK:      " << bermudanSwaption.NPV() << std::endl;
272
273
274     // OTM Bermudan swaption pricing
275
276     std::cout << "Payer bermudan swaption "
277         << "struck at " << io::rate(fixedOTMRate)
278         << " (OTM)" << std::endl;
279
280     Swaption otmBermudanSwaption(otmSwap, bermudanExercise, rhTermStructure,
281         boost::shared_ptr<PricingEngine>());

```

```

282
283 // Do the pricing for each model
284 otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
285     new TreeSwaptionEngine(modelG2, 50)));
286 std::cout << "G2:      " << otmBermudanSwaption.NPV() << std::endl;
287
288 otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
289     new TreeSwaptionEngine(modelHW, 50)));
290 std::cout << "HW:      " << otmBermudanSwaption.NPV() << std::endl;
291
292 otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
293     new TreeSwaptionEngine(modelHW2, 50)));
294 std::cout << "HW (num): " << otmBermudanSwaption.NPV() << std::endl;
295
296 otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
297     new TreeSwaptionEngine(modelBK, 50)));
298 std::cout << "BK:      " << otmBermudanSwaption.NPV() << std::endl;
299
300
301 // ITM Bermudan swaption pricing
302
303 std::cout << "Payer bermudan swaption "
304     << "struck at " << io::rate(fixedITMRate)
305     << " (ITM)" << std::endl;
306
307 Swaption itmBermudanSwaption(itmSwap,bermudanExercise,rhTermStructure,
308     boost::shared_ptr<PricingEngine>());
309
310 // Do the pricing for each model
311 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
312     new TreeSwaptionEngine(modelG2, 50)));
313 std::cout << "G2:      " << itmBermudanSwaption.NPV() << std::endl;
314
315 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
316     new TreeSwaptionEngine(modelHW, 50)));
317 std::cout << "HW:      " << itmBermudanSwaption.NPV() << std::endl;
318
319 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
320     new TreeSwaptionEngine(modelHW2, 50)));
321 std::cout << "HW (num): " << itmBermudanSwaption.NPV() << std::endl;
322
323 itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
324     new TreeSwaptionEngine(modelBK, 50)));
325 std::cout << "BK:      " << itmBermudanSwaption.NPV() << std::endl;
326
327 return 0;
328 } catch (std::exception& e) {
329     std::cout << e.what() << std::endl;
330     return 1;
331 } catch (...) {
332     std::cout << "unknown error" << std::endl;
333     return 1;
334 }
335 }
336

```

9.3 DiscreteHedging.cpp

This is an example of using the QuantLib Monte Carlo framework.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
20 /* This example computes profit and loss of a discrete interval hedging
21    strategy and compares with the results of Derman & Kamal's (Goldman Sachs
22    Equity Derivatives Research) Research Note: "When You Cannot Hedge
23    Continuously: The Corrections to Black-Scholes"
24    http://www.ederman.com/emanuelderman/GSQSpapers/when_you_cannot_hedge.pdf
25
26    Suppose an option hedger sells an European option and receives the
27    Black-Scholes value as the options premium.
28    Then he follows a Black-Scholes hedging strategy, rehedging at discrete,
29    evenly spaced time intervals as the underlying stock changes. At
30    expiration, the hedger delivers the option payoff to the option holder,
31    and unwinds the hedge. We are interested in understanding the final
32    profit or loss of this strategy.
33
34    If the hedger had followed the exact Black-Scholes replication strategy,
35    re-hedging continuously as the underlying stock evolved towards its final
36    value at expiration, then, no matter what path the stock took, the final
37    P&L would be exactly zero. When the replication strategy deviates from
38    the exact Black-Scholes method, the final P&L may deviate from zero. This
39    deviation is called the replication error. When the hedger rebalances at
40    discrete rather than continuous intervals, the hedge is imperfect and the
41    replication is inexact. The more often hedging occurs, the smaller the
42    replication error.
43
44    We examine the range of possibilities, computing the replication error.
45  */
46
47 // the only header you need to use QuantLib
48 #include <ql/quantlib.hpp>
49 #include <iostream>
50
51 using namespace QuantLib;
52
53 #if defined(QL_ENABLE_SESSIONS)
54 namespace QuantLib {
55     Integer sessionId() { return 0; }
56 }
57 #endif
58
59
60
61
62 /* The ReplicationError class carries out Monte Carlo simulations to evaluate
63    the outcome (the replication error) of the discrete hedging strategy over
64    different, randomly generated scenarios of future stock price evolution.
65  */
66 class ReplicationError
67 {
68 public:
69     ReplicationError(Option::Type type,
70                     Time maturity,
71                     Real strike,
72                     Real s0,
73                     Volatility sigma,
74                     Rate r)
75     : maturity_(maturity), payoff_(type, strike), s0_(s0),
76       sigma_(sigma), r_(r) {
77
78         // value of the option
79         DiscountFactor rDiscount = std::exp(-r_*maturity_);

```

```

80     DiscountFactor qDiscount = 1.0;
81     Real forward = s0_*qDiscount/rDiscount;
82     Real variance = sigma_*sigma_*maturity_;
83     boost::shared_ptr<StrikedTypePayoff> payoff(
84         new PlainVanillaPayoff(payoff_));
85     BlackFormula black(forward,rDiscount,variance,payoff);
86     std::cout << "Option value: " << black.value() << std::endl;
87
88     // store option's vega, since Derman and Kamal's formula needs it
89     vega_ = black.vega(maturity_);
90
91     std::cout << std::endl;
92     std::cout <<
93         "          |          | P&L  \t|  P&L      | Derman&Kamal | P&L"
94         "          \t| P&L" << std::endl;
95
96     std::cout <<
97         "samples | trades | Mean \t| Std Dev | Formula      |"
98         " skewness \t| kurt." << std::endl;
99
100    std::cout << "-----"
101        "-----" << std::endl;
102    }
103
104    // the actual replication error computation
105    void compute(Size nTimeSteps, Size nSamples);
106 private:
107     Time maturity_;
108     PlainVanillaPayoff payoff_;
109     Real s0_;
110     Volatility sigma_;
111     Rate r_;
112     Real vega_;
113 };
114
115 // The key for the MonteCarlo simulation is to have a PathPricer that
116 // implements a value(const Path& path) method.
117 // This method prices the portfolio for each Path of the random variable
118 class ReplicationPathPricer : public PathPricer<Path> {
119 public:
120     // real constructor
121     ReplicationPathPricer(Option::Type type,
122                          Real underlying,
123                          Real strike,
124                          Rate r,
125                          Time maturity,
126                          Volatility sigma)
127     : type_(type), underlying_(underlying),
128       strike_(strike), r_(r), maturity_(maturity), sigma_(sigma) {
129         QL_REQUIRE(strike_ > 0.0, "strike must be positive");
130         QL_REQUIRE(underlying_ > 0.0, "underlying 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 underlying_, 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         Time maturity = 1.0/12.0;    // 1 month
157         Real strike = 100;
158         Real underlying = 100;
159         Volatility volatility = 0.20; // 20%
160         Rate riskFreeRate = 0.05; // 5%
161         ReplicationError rp(Option::Call, maturity, strike, underlying,
162                             volatility, riskFreeRate);
163
164         Size scenarios = 50000;
165         Size hedgesNum;
166
167         hedgesNum = 21;
168         rp.compute(hedgesNum, scenarios);
169
170         hedgesNum = 84;
171         rp.compute(hedgesNum, scenarios);
172
173         return 0;
174     } catch (std::exception& e) {
175         std::cout << e.what() << std::endl;
176         return 1;
177     } catch (...) {
178         std::cout << "unknown error" << std::endl;
179         return 1;
180     }
181 }
182
183
184 /* The actual computation of the Profit&Loss for each single path.
185
186     In each scenario N reheding trades spaced evenly in time over
187     the life of the option are carried out, using the Black-Scholes
188     hedge ratio.
189 */
190 Real ReplicationPathPricer::operator()(const Path& path) const {
191
192     Size n = path.length()-1;
193     QL_REQUIRE(n>0, "the path cannot be empty");
194
195     // discrete hedging interval
196     Time dt = maturity_/n;
197
198     // For simplicity, we assume the stock pays no dividends.
199     Rate stockDividendYield = 0.0;
200
201     // let's start
202     Time t = 0;
203
204     // stock value at t=0
205     Real stock = underlying_;
206
207     // money account at t=0
208     Real money_account = 0.0;
209
210     /***** the initial deal *****/
211     /***** option fair price (Black-Scholes) at t=0

```

```

214 DiscountFactor rDiscount = std::exp(-r_*maturity_);
215 DiscountFactor qDiscount = std::exp(-stockDividendYield*maturity_);
216 Real forward = stock*qDiscount/rDiscount;
217 Real variance = sigma_*sigma_*maturity_;
218 boost::shared_ptr<StrikedTypePayoff> payoff(
219     new PlainVanillaPayoff(type_,strike_));
220 BlackFormula black(forward,rDiscount,variance,payoff);
221 // sell the option, cash in its premium
222 money_account += black.value();
223 // compute delta
224 Real delta = black.delta(stock);
225 // delta-hedge the option buying stock
226 Real stockAmount = delta;
227 money_account -= stockAmount*stock;
228
229 /*****/
230 /** hedging during option life */
231 /*****/
232 for (Size step = 0; step < n-1; step++){
233
234     // time flows
235     t += dt;
236
237     // accruing on the money account
238     money_account *= std::exp( r_*dt );
239
240     // stock growth:
241     stock = path[step+1];
242
243     // recalculate option value at the current stock value,
244     // and the current time to maturity
245     rDiscount = std::exp(-r_*(maturity_-t));
246     qDiscount = std::exp(-stockDividendYield*(maturity_-t));
247     forward = stock*qDiscount/rDiscount;
248     variance = sigma_*sigma_*(maturity_-t);
249     BlackFormula black(forward,rDiscount,variance,payoff);
250
251     // recalculate delta
252     delta = black.delta(stock);
253
254     // re-hedging
255     money_account -= (delta - stockAmount)*stock;
256     stockAmount = delta;
257 }
258
259 /*****/
260 /** option expiration */
261 /*****/
262 // last accrual on my money account
263 money_account *= std::exp( r_*dt );
264 // last stock growth
265 stock = path[n];
266
267 // the hedger delivers the option payoff to the option holder
268 Real optionPayoff = PlainVanillaPayoff(type_, strike_)(stock);
269 money_account -= optionPayoff;
270
271 // and unwinds the hedge selling his stock position
272 money_account += stockAmount*stock;
273
274 // final Profit&Loss
275 return money_account;
276 }
277
278
279 // The computation over nSamples paths of the P&L distribution
280 void ReplicationError::compute(Size nTimeSteps, Size nSamples)

```

```

281 {
282     QL_REQUIRE(nTimeSteps>0, "the number of steps must be > 0");
283
284     // hedging interval
285     // Time tau = maturity_ / nTimeSteps;
286
287     /* Black-Scholes framework: the underlying stock price evolves
288        lognormally with a fixed known volatility that stays constant
289        throughout time.
290     */
291     Date today = Date::todaysDate();
292     DayCounter dayCount = Actual365Fixed();
293     Handle<Quote> stateVariable(
294         boost::shared_ptr<Quote>(new SimpleQuote(s0_)));
295     Handle<YieldTermStructure> riskFreeRate(
296         boost::shared_ptr<YieldTermStructure>(
297             new FlatForward(today, r_, dayCount)));
298     Handle<YieldTermStructure> dividendYield(
299         boost::shared_ptr<YieldTermStructure>(
300             new FlatForward(today, 0.0, dayCount)));
301     Handle<BlackVolTermStructure> volatility(
302         boost::shared_ptr<BlackVolTermStructure>(
303             new BlackConstantVol(today, sigma_, dayCount)));
304     boost::shared_ptr<StochasticProcess1D> diffusion(
305         new BlackScholesProcess(stateVariable, dividendYield,
306             riskFreeRate, volatility));
307
308     // Black Scholes equation rules the path generator:
309     // at each step the log of the stock
310     // will have drift and sigma^2 variance
311     PseudoRandom::rsg_type rsg =
312         PseudoRandom::make_sequence_generator(nTimeSteps, 0);
313
314     bool brownianBridge = false;
315
316     typedef SingleVariate<PseudoRandom>::path_generator_type generator_type;
317     boost::shared_ptr<generator_type> myPathGenerator(new
318         generator_type(diffusion, maturity_, nTimeSteps,
319             rsg, brownianBridge));
320
321     // The replication strategy's Profit&Loss is computed for each path
322     // of the stock. The path pricer knows how to price a path using its
323     // value() method
324     boost::shared_ptr<PathPricer<Path> > myPathPricer(new
325         ReplicationPathPricer(payoff_.optionType(), s0_,
326             payoff_.strike(), r_, maturity_, sigma_));
327
328     // a statistics accumulator for the path-dependant Profit&Loss values
329     Statistics statisticsAccumulator;
330
331     // The OneFactorMonteCarloModel generates paths using myPathGenerator
332     // each path is priced using myPathPricer
333     // prices will be accumulated into statisticsAccumulator
334     OneFactorMonteCarloOption MCSimulation(myPathGenerator,
335         myPathPricer,
336         statisticsAccumulator,
337         false);
338
339     // the model simulates nSamples paths
340     MCSimulation.addSamples(nSamples);
341
342     // the sampleAccumulator method of OneFactorMonteCarloOption_old
343     // gives access to all the methods of statisticsAccumulator
344     Real PLMean = MCSimulation.sampleAccumulator().mean();
345     Real PLStDev = MCSimulation.sampleAccumulator().standardDeviation();
346     Real PLSkew = MCSimulation.sampleAccumulator().skewness();
347     Real PLKurt = MCSimulation.sampleAccumulator().kurtosis();

```

```
348
349 // Derman and Kamal's formula
350 Real theorStd = std::sqrt(M_PI/4/nTimeSteps)*vega_*sigma_;
351
352
353 std::cout << std::fixed
354           << nSamples << "\t| "
355           << nTimeSteps << "\t | "
356           << std::setprecision(3) << PLMean << " \t| "
357           << std::setprecision(2) << PLStDev << " \t | "
358           << std::setprecision(2) << theorStd << " \t | "
359           << std::setprecision(2) << PLSkew << " \t| "
360           << std::setprecision(2) << PLKurt << std::endl;
361 }
```


9.4 EuropeanOption.cpp

This example calculates European options using different methods while testing call-put parity.

```

1  /* -*- mode: c++; tab-width: 4; indent-tabs-mode: nil; c-basic-offset: 4 -*- */
2
21 #include <ql/quantlib.hpp>
22 #include <iostream>
23
24 using namespace QuantLib;
25
26 #if defined(QL_ENABLE_SESSIONS)
27 namespace QuantLib {
28
29     Integer sessionId() { return 0; }
30
31 }
32 #endif
33
34
35 // This will be included in the library after a bit of redesign
36 class WeightedPayoff {
37     public:
38         WeightedPayoff(Option::Type type,
39             Time maturity,
40             Real strike,
41             Real s0,
42             Volatility sigma,
43             Rate r,
44             Rate q)
45             : type_(type), maturity_(maturity),
46               strike_(strike),
47               s0_(s0),
48               sigma_(sigma), r_(r), q_(q) {}
49
50         Real operator()(Real x) const {
51             Real nuT = (r_-q_-0.5*sigma_*sigma_)*maturity_;
52             return std::exp(-r_*maturity_)
53                 *PlainVanillaPayoff(type_, strike_)(s0_*std::exp(x))
54                 *std::exp(-(x - nuT)*(x - nuT)/(2*sigma_*sigma_*maturity_))
55                 /std::sqrt(2.0*M_PI*sigma_*sigma_*maturity_);
56         }
57     private:
58         Option::Type type_;
59         Time maturity_;
60         Real strike_;
61         Real s0_;
62         Volatility sigma_;
63         Rate r_,q_;
64 };
65
66
67 int main(int, char* [])
68 {
69     try {
70         QL_IO_INIT
71
72         std::cout << "Using " << QL_VERSION << std::endl << std::endl;
73
74         // our option
75         Option::Type type(Option::Call);
76         Real underlying = 7;
77         Real strike = 8;
78         Spread dividendYield = 0.05;
79         Rate riskFreeRate = 0.05;
80

```

```

81     Date todaysDate(15, May, 1998);
82     Date settlementDate(17, May, 1998);
83     Settings::instance().evaluationDate() = todaysDate;
84
85     Date exerciseDate(17, May, 1999);
86     DayCounter dayCounter = Actual365Fixed();
87     Time maturity = dayCounter.yearFraction(settlementDate,
88                                           exerciseDate);
89
90     Volatility volatility = 0.10;
91     std::cout << "option type = " << type << std::endl;
92     std::cout << "Time to maturity = " << maturity
93               << std::endl;
94     std::cout << "Underlying price = " << underlying
95               << std::endl;
96     std::cout << "Strike = " << strike
97               << std::endl;
98     std::cout << "Risk-free interest rate = " << io::rate(riskFreeRate)
99               << std::endl;
100    std::cout << "Dividend yield = " << io::rate(dividendYield)
101              << std::endl;
102    std::cout << "Volatility = " << io::volatility(volatility)
103              << std::endl;
104    std::cout << std::endl;
105
106    Date midlifeDate(19, November, 1998);
107    std::vector<Date> exDates(2);
108    exDates[0]=midlifeDate;
109    exDates[1]=exerciseDate;
110
111    boost::shared_ptr<Exercise> exercise(
112        new EuropeanExercise(exerciseDate));
113    boost::shared_ptr<Exercise> amExercise(
114        new AmericanExercise(settlementDate,
115                             exerciseDate));
116    boost::shared_ptr<Exercise> berExercise(new BermudanExercise(exDates));
117
118
119    Handle<Quote> underlyingH(
120        boost::shared_ptr<Quote>(new SimpleQuote(underlying)));
121
122    // bootstrap the yield/dividend/vol curves
123    Handle<YieldTermStructure> flatTermStructure(
124        boost::shared_ptr<YieldTermStructure>(
125            new FlatForward(settlementDate, riskFreeRate, dayCounter)));
126    Handle<YieldTermStructure> flatDividendTS(
127        boost::shared_ptr<YieldTermStructure>(
128            new FlatForward(settlementDate, dividendYield, dayCounter)));
129    Handle<BlackVolTermStructure> flatVolTS(
130        boost::shared_ptr<BlackVolTermStructure>(
131            new BlackConstantVol(settlementDate, volatility, dayCounter)));
132
133    std::vector<Date> dates(4);
134    dates[0] = settlementDate + 1*Months;
135    dates[1] = exerciseDate;
136    dates[2] = exerciseDate + 6*Months;
137    dates[3] = exerciseDate + 12*Months;
138    std::vector<Real> strikes(4);
139    strikes[0] = underlying*0.9;
140    strikes[1] = underlying;
141    strikes[2] = underlying*1.1;
142    strikes[3] = underlying*1.2;
143
144    Matrix vols(4,4);
145    vols[0][0] = volatility*1.1;
146    vols[0][1] = volatility;
147    vols[0][2] = volatility*0.9;

```

```

148                                     vols[0][3] = volatility*0.8;
149     vols[1][0] = volatility*1.1;
150         vols[1][1] = volatility;
151         vols[1][2] = volatility*0.9;
152         vols[1][3] = volatility*0.8;
153     vols[2][0] = volatility*1.1;
154         vols[2][1] = volatility;
155         vols[2][2] = volatility*0.9;
156         vols[2][3] = volatility*0.8;
157     vols[3][0] = volatility*1.1;
158         vols[3][1] = volatility;
159         vols[3][2] = volatility*0.9;
160         vols[3][3] = volatility*0.8;
161
162     Handle<BlackVolTermStructure> blackSurface(
163         boost::shared_ptr<BlackVolTermStructure>(
164             new BlackVarianceSurface(settlementDate, dates,
165                                     strikes, vols, dayCounter)));
166
167
168     boost::shared_ptr<StrikedTypePayoff> payoff(new
169         PlainVanillaPayoff(type, strike));
170
171     boost::shared_ptr<BlackScholesProcess> stochasticProcess(new
172         BlackScholesProcess(underlyingH, flatDividendTS,
173                             flatTermStructure,
174                             // blackSurface
175                             flatVolTS));
176
177     EuropeanOption option(stochasticProcess, payoff, exercise);
178
179
180     std::string method;
181     Real value, discrepancy, rightValue, relativeDiscrepancy;
182
183     std::cout << std::endl << std::endl;
184
185     // write column headings
186     std::cout << "Method\t\tValue\t\tEstimatedError\tDiscrepancy"
187         "\tRel. Discr." << std::endl;
188
189     // method: Black-Scholes Engine
190     method = "Black-Scholes";
191     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
192         new AnalyticEuropeanEngine()));
193     rightValue = value = option.NPV();
194     discrepancy = std::fabs(value-rightValue);
195     relativeDiscrepancy = discrepancy/rightValue;
196     std::cout << method << "\t"
197         << value << "\t" << "N/A\t\t"
198         << discrepancy << "\t\t" << relativeDiscrepancy << std::endl;
199
200
201     // method: Integral
202     method = "Integral";
203     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
204         new IntegralEngine()));
205     value = option.NPV();
206     discrepancy = std::fabs(value-rightValue);
207     relativeDiscrepancy = discrepancy/rightValue;
208     std::cout << method << "\t"
209         << value << "\t" << "N/A\t\t"
210         << discrepancy << "\t\t" << relativeDiscrepancy << std::endl;
211
212 /*
213     // method: Integral
214     method = "Binary Cash";

```

```

215     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
216         new IntegralCashOrNothingEngine(1.0)));
217     value = option.NPV();
218     discrepancy = std::fabs(value-rightValue);
219     relativeDiscrepancy = discrepancy/rightValue;
220     std::cout << method << "\t"
221         << value << "\t" << "N/A\t\t"
222         << discrepancy << "\t" << relativeDiscrepancy << std::endl;
223
224     // method: Integral
225     method = "Binary Asset";
226     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
227         new IntegralAssetOrNothingEngine()));
228     value = option.NPV();
229     discrepancy = std::fabs(value-rightValue);
230     relativeDiscrepancy = discrepancy/rightValue;
231     std::cout << method << "\t"
232         << value << "\t" << "N/A\t\t"
233         << discrepancy << "\t" << relativeDiscrepancy << std::endl;
234
235 */
236     Size timeSteps = 801;
237
238     // Binomial Method (JR)
239     method = "Binomial (JR)";
240     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
241         new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
242     value = option.NPV();
243     discrepancy = std::fabs(value-rightValue);
244     relativeDiscrepancy = discrepancy/rightValue;
245     std::cout << method << "\t"
246         << value << "\t" << "N/A\t\t"
247         << discrepancy << "\t" << relativeDiscrepancy << std::endl;
248
249
250     // Binomial Method (CRR)
251     method = "Binomial (CRR)";
252     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
253         new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
254     value = option.NPV();
255     discrepancy = std::fabs(value-rightValue);
256     relativeDiscrepancy = discrepancy/rightValue;
257     std::cout << method << "\t"
258         << value << "\t" << "N/A\t\t"
259         << discrepancy << "\t" << relativeDiscrepancy << std::endl;
260
261     // Equal Probability Additive Binomial Tree (EQP)
262     method = "Additive (EQP)";
263     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
264         new BinomialVanillaEngine<AdditiveEQPBinomialTree>(timeSteps)));
265     value = option.NPV();
266     discrepancy = std::fabs(value-rightValue);
267     relativeDiscrepancy = discrepancy/rightValue;
268     std::cout << method << "\t"
269         << value << "\t" << "N/A\t\t"
270         << discrepancy << "\t" << relativeDiscrepancy << std::endl;
271
272     // Equal Jumps Additive Binomial Tree (Trigeorgis)
273     method = "Bin. Trigeorgis";
274     option.setPricingEngine(boost::shared_ptr<PricingEngine>(
275         new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
276     value = option.NPV();
277     discrepancy = std::fabs(value-rightValue);
278     relativeDiscrepancy = discrepancy/rightValue;
279     std::cout << method << "\t"
280         << value << "\t" << "N/A\t\t"
281         << discrepancy << "\t" << relativeDiscrepancy << std::endl;

```

```

282
283 // Tian Binomial Tree (third moment matching)
284 method = "Binomial Tian";
285 option.setPricingEngine(boost::shared_ptr<PricingEngine>(
286     new BinomialVanillaEngine<Tian>(timeSteps)));
287 value = option.NPV();
288 discrepancy = std::fabs(value-rightValue);
289 relativeDiscrepancy = discrepancy/rightValue;
290 std::cout << method << "\t"
291     << value << "\t" << "N/A\t\t"
292     << discrepancy << "\t" << relativeDiscrepancy << std::endl;
293
294 // Leisen-Reimer Binomial Tree
295 method = "Binomial LR";
296 option.setPricingEngine(boost::shared_ptr<PricingEngine>(
297     new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
298 value = option.NPV();
299 discrepancy = std::fabs(value-rightValue);
300 relativeDiscrepancy = discrepancy/rightValue;
301 std::cout << method << "\t"
302     << value << "\t" << "N/A\t\t"
303     << discrepancy << "\t" << relativeDiscrepancy << std::endl;
304
305 // Finite Differences
306
307 method = "Finite Diff.";
308 timeSteps = 100;
309 Size gridPoints = 100;
310 option.setPricingEngine(boost::shared_ptr<PricingEngine>(
311     new FDEuropeanEngine(timeSteps, gridPoints)));
312 value = option.NPV();
313 discrepancy = std::fabs(value-rightValue);
314 relativeDiscrepancy = discrepancy/rightValue;
315 std::cout << method << "\t"
316     << value << "\t" << "N/A\t\t"
317     << discrepancy << "\t" << relativeDiscrepancy << std::endl;
318
319 // Monte Carlo Method
320 timeSteps = 1;
321
322 method = "MC (crude)";
323 Size mcSeed = 42;
324
325 boost::shared_ptr<PricingEngine> mcengine1;
326 mcengine1 =
327     MakeMCEuropeanEngine<PseudoRandom>().withSteps(timeSteps)
328     .withTolerance(0.02)
329     .withSeed(mcSeed);
330 option.setPricingEngine(mcengine1);
331
332 value = option.NPV();
333 Real errorEstimate = option.errorEstimate();
334 discrepancy = std::fabs(value-rightValue);
335 relativeDiscrepancy = discrepancy/rightValue;
336 std::cout << method << "\t"
337     << value << "\t" << errorEstimate << "\t"
338     << discrepancy << "\t" << relativeDiscrepancy << std::endl;
339
340 method = "MC (Sobol)";
341 timeSteps = 1;
342 Size nSamples = 32768; // 2^15
343
344 boost::shared_ptr<PricingEngine> mcengine2;
345 mcengine2 =
346     MakeMCEuropeanEngine<LowDiscrepancy>().withSteps(timeSteps)
347     .withSamples(nSamples);
348 option.setPricingEngine(mcengine2);

```

```
349
350     value = option.NPV();
351     discrepancy = std::fabs(value-rightValue);
352     relativeDiscrepancy = discrepancy/rightValue;
353     std::cout << method << "\t"
354               << value << "\t" << "N/A\t\t"
355               << discrepancy << "\t" << relativeDiscrepancy << std::endl;
356
357     return 0;
358 } catch (std::exception& e) {
359     std::cout << e.what() << std::endl;
360     return 1;
361 } catch (...) {
362     std::cout << "unknown error" << std::endl;
363     return 1;
364 }
365 }
```

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 <iostream>
28 #include <iomanip>
29
30 using namespace QuantLib;
31
32 #if defined(QL_ENABLE_SESSIONS)
33 namespace QuantLib {
34
35     Integer sessionId() { return 0; }
36
37 }
38 #endif
39
40
41 int main(int, char* [])
42 {
43     try {
44         QL_IO_INIT
45
46         /******
47          *** MARKET DATA ***
48          *****/
49
50         Calendar calendar = TARGET();
51         // uncommenting the following line generates an error
52         // calendar = Tokyo();
53         Date settlementDate(22, September, 2004);
54         // must be a business day
55         settlementDate = calendar.adjust(settlementDate);
56
57         Integer fixingDays = 2;
58         Date todaysDate = calendar.advance(settlementDate, -fixingDays, Days);
59         // nothing to do with Date::todaysDate
60         Settings::instance().evaluationDate() = todaysDate;
61
62
63         todaysDate = Settings::instance().evaluationDate();
64         std::cout << "Today: " << todaysDate.weekday()
65                   << ", " << todaysDate << std::endl;
66
67         std::cout << "Settlement date: " << settlementDate.weekday()
68                   << ", " << settlementDate << std::endl;
69
70         // deposits
71         Rate d1wQuote=0.0382;
72         Rate d1mQuote=0.0372;
73         Rate d3mQuote=0.0363;
74         Rate d6mQuote=0.0353;
75         Rate d9mQuote=0.0348;
76         Rate d1yQuote=0.0345;
77         // FRAs
78         Rate fra3x6Quote=0.037125;
79         Rate fra6x9Quote=0.037125;
80         Rate fra6x12Quote=0.037125;

```



```

81     // futures
82     Real fut1Quote=96.2875;
83     Real fut2Quote=96.7875;
84     Real fut3Quote=96.9875;
85     Real fut4Quote=96.6875;
86     Real fut5Quote=96.4875;
87     Real fut6Quote=96.3875;
88     Real fut7Quote=96.2875;
89     Real fut8Quote=96.0875;
90     // swaps
91     Rate s2yQuote=0.037125;
92     Rate s3yQuote=0.0398;
93     Rate s5yQuote=0.0443;
94     Rate s10yQuote=0.05165;
95     Rate s15yQuote=0.055175;
96
97
98     /*****
99     ***   QUOTES   ***
100    *****/
101
102    // SimpleQuote stores a value which can be manually changed;
103    // other Quote subclasses could read the value from a database
104    // or some kind of data feed.
105
106    // deposits
107    boost::shared_ptr<Quote> dlwRate(new SimpleQuote(dlwQuote));
108    boost::shared_ptr<Quote> d1mRate(new SimpleQuote(d1mQuote));
109    boost::shared_ptr<Quote> d3mRate(new SimpleQuote(d3mQuote));
110    boost::shared_ptr<Quote> d6mRate(new SimpleQuote(d6mQuote));
111    boost::shared_ptr<Quote> d9mRate(new SimpleQuote(d9mQuote));
112    boost::shared_ptr<Quote> d1yRate(new SimpleQuote(d1yQuote));
113    // FRAs
114    boost::shared_ptr<Quote> fra3x6Rate(new SimpleQuote(fra3x6Quote));
115    boost::shared_ptr<Quote> fra6x9Rate(new SimpleQuote(fra6x9Quote));
116    boost::shared_ptr<Quote> fra6x12Rate(new SimpleQuote(fra6x12Quote));
117    // futures
118    boost::shared_ptr<Quote> fut1Price(new SimpleQuote(fut1Quote));
119    boost::shared_ptr<Quote> fut2Price(new SimpleQuote(fut2Quote));
120    boost::shared_ptr<Quote> fut3Price(new SimpleQuote(fut3Quote));
121    boost::shared_ptr<Quote> fut4Price(new SimpleQuote(fut4Quote));
122    boost::shared_ptr<Quote> fut5Price(new SimpleQuote(fut5Quote));
123    boost::shared_ptr<Quote> fut6Price(new SimpleQuote(fut6Quote));
124    boost::shared_ptr<Quote> fut7Price(new SimpleQuote(fut7Quote));
125    boost::shared_ptr<Quote> fut8Price(new SimpleQuote(fut8Quote));
126    // swaps
127    boost::shared_ptr<Quote> s2yRate(new SimpleQuote(s2yQuote));
128    boost::shared_ptr<Quote> s3yRate(new SimpleQuote(s3yQuote));
129    boost::shared_ptr<Quote> s5yRate(new SimpleQuote(s5yQuote));
130    boost::shared_ptr<Quote> s10yRate(new SimpleQuote(s10yQuote));
131    boost::shared_ptr<Quote> s15yRate(new SimpleQuote(s15yQuote));
132
133
134    /*****
135    ***   RATE HELPERS   ***
136    *****/
137
138    // RateHelpers are built from the above quotes together with
139    // other instrument dependant infos. Quotes are passed in
140    // relinkable handles which could be relinked to some other
141    // data source later.
142
143    // deposits
144    DayCounter depositDayCounter = Actual360();
145
146    boost::shared_ptr<RateHelper> dlw(new DepositRateHelper(
147        Handle<Quote>(dlwRate),

```

```

148         1, Weeks, fixingDays,
149         calendar, ModifiedFollowing, depositDayCounter));
150     boost::shared_ptr<RateHelper> d1m(new DepositRateHelper(
151         Handle<Quote>(d1mRate),
152         1, Months, fixingDays,
153         calendar, ModifiedFollowing, depositDayCounter));
154     boost::shared_ptr<RateHelper> d3m(new DepositRateHelper(
155         Handle<Quote>(d3mRate),
156         3, Months, fixingDays,
157         calendar, ModifiedFollowing, depositDayCounter));
158     boost::shared_ptr<RateHelper> d6m(new DepositRateHelper(
159         Handle<Quote>(d6mRate),
160         6, Months, fixingDays,
161         calendar, ModifiedFollowing, depositDayCounter));
162     boost::shared_ptr<RateHelper> d9m(new DepositRateHelper(
163         Handle<Quote>(d9mRate),
164         9, Months, fixingDays,
165         calendar, ModifiedFollowing, depositDayCounter));
166     boost::shared_ptr<RateHelper> d1y(new DepositRateHelper(
167         Handle<Quote>(d1yRate),
168         1, Years, fixingDays,
169         calendar, ModifiedFollowing, depositDayCounter));
170
171
172     // setup FRAs
173     boost::shared_ptr<RateHelper> fra3x6(new FraRateHelper(
174         Handle<Quote>(fra3x6Rate),
175         3, 6, fixingDays, calendar, ModifiedFollowing,
176         depositDayCounter));
177     boost::shared_ptr<RateHelper> fra6x9(new FraRateHelper(
178         Handle<Quote>(fra6x9Rate),
179         6, 9, fixingDays, calendar, ModifiedFollowing,
180         depositDayCounter));
181     boost::shared_ptr<RateHelper> fra6x12(new FraRateHelper(
182         Handle<Quote>(fra6x12Rate),
183         6, 12, fixingDays, calendar, ModifiedFollowing,
184         depositDayCounter));
185
186
187     // setup futures
188     Integer futMonths = 3;
189     Date imm = Date::nextIMMdate(settlementDate);
190     boost::shared_ptr<RateHelper> fut1(new FuturesRateHelper(
191         Handle<Quote>(fut1Price),
192         imm,
193         futMonths, calendar, ModifiedFollowing,
194         depositDayCounter));
195     imm = Date::nextIMMdate(imm+1);
196     boost::shared_ptr<RateHelper> fut2(new FuturesRateHelper(
197         Handle<Quote>(fut1Price),
198         imm,
199         futMonths, calendar, ModifiedFollowing,
200         depositDayCounter));
201     imm = Date::nextIMMdate(imm+1);
202     boost::shared_ptr<RateHelper> fut3(new FuturesRateHelper(
203         Handle<Quote>(fut1Price),
204         imm,
205         futMonths, calendar, ModifiedFollowing,
206         depositDayCounter));
207     imm = Date::nextIMMdate(imm+1);
208     boost::shared_ptr<RateHelper> fut4(new FuturesRateHelper(
209         Handle<Quote>(fut1Price),
210         imm,
211         futMonths, calendar, ModifiedFollowing,
212         depositDayCounter));
213     imm = Date::nextIMMdate(imm+1);
214     boost::shared_ptr<RateHelper> fut5(new FuturesRateHelper(

```

```

215         Handle<Quote>(fut1Price),
216         imm,
217         futMonths, calendar, ModifiedFollowing,
218         depositDayCounter));
219     imm = Date::nextIMMdate(imm+1);
220     boost::shared_ptr<RateHelper> fut6(new FuturesRateHelper(
221         Handle<Quote>(fut1Price),
222         imm,
223         futMonths, calendar, ModifiedFollowing,
224         depositDayCounter));
225     imm = Date::nextIMMdate(imm+1);
226     boost::shared_ptr<RateHelper> fut7(new FuturesRateHelper(
227         Handle<Quote>(fut1Price),
228         imm,
229         futMonths, calendar, ModifiedFollowing,
230         depositDayCounter));
231     imm = Date::nextIMMdate(imm+1);
232     boost::shared_ptr<RateHelper> fut8(new FuturesRateHelper(
233         Handle<Quote>(fut1Price),
234         imm,
235         futMonths, calendar, ModifiedFollowing,
236         depositDayCounter));
237
238
239     // setup swaps
240     Frequency swFixedLegFrequency = Annual;
241     BusinessDayConvention swFixedLegConvention = Unadjusted;
242     DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
243     Frequency swFloatingLegFrequency = Semiannual;
244
245     boost::shared_ptr<RateHelper> s2y(new SwapRateHelper(
246         Handle<Quote>(s2yRate),
247         2, Years, fixingDays,
248         calendar, swFixedLegFrequency,
249         swFixedLegConvention, swFixedLegDayCounter,
250         swFloatingLegFrequency, ModifiedFollowing));
251     boost::shared_ptr<RateHelper> s3y(new SwapRateHelper(
252         Handle<Quote>(s3yRate),
253         3, Years, fixingDays,
254         calendar, swFixedLegFrequency,
255         swFixedLegConvention, swFixedLegDayCounter,
256         swFloatingLegFrequency, ModifiedFollowing));
257     boost::shared_ptr<RateHelper> s5y(new SwapRateHelper(
258         Handle<Quote>(s5yRate),
259         5, Years, fixingDays,
260         calendar, swFixedLegFrequency,
261         swFixedLegConvention, swFixedLegDayCounter,
262         swFloatingLegFrequency, ModifiedFollowing));
263     boost::shared_ptr<RateHelper> s10y(new SwapRateHelper(
264         Handle<Quote>(s10yRate),
265         10, Years, fixingDays,
266         calendar, swFixedLegFrequency,
267         swFixedLegConvention, swFixedLegDayCounter,
268         swFloatingLegFrequency, ModifiedFollowing));
269     boost::shared_ptr<RateHelper> s15y(new SwapRateHelper(
270         Handle<Quote>(s15yRate),
271         15, Years, fixingDays,
272         calendar, swFixedLegFrequency,
273         swFixedLegConvention, swFixedLegDayCounter,
274         swFloatingLegFrequency, ModifiedFollowing));
275
276
277     /*****
278     ** CURVE BUILDING **
279     *****/
280
281     // Any DayCounter would be fine.

```

```

282 // ActualActual::ISDA ensures that 30 years is 30.0
283 DayCounter termStructureDayCounter =
284     ActualActual(ActualActual::ISDA);
285
286
287 double tolerance = 1.0e-15;
288
289 // A depo-swap curve
290 std::vector<boost::shared_ptr<RateHelper> > depoSwapInstruments;
291 depoSwapInstruments.push_back(d1w);
292 depoSwapInstruments.push_back(d1m);
293 depoSwapInstruments.push_back(d3m);
294 depoSwapInstruments.push_back(d6m);
295 depoSwapInstruments.push_back(d9m);
296 depoSwapInstruments.push_back(d1y);
297 depoSwapInstruments.push_back(s2y);
298 depoSwapInstruments.push_back(s3y);
299 depoSwapInstruments.push_back(s5y);
300 depoSwapInstruments.push_back(s10y);
301 depoSwapInstruments.push_back(s15y);
302 boost::shared_ptr<YieldTermStructure> depoSwapTermStructure(new
303     PiecewiseFlatForward(settlementDate, depoSwapInstruments,
304         termStructureDayCounter, tolerance));
305
306
307 // A depo-futures-swap curve
308 std::vector<boost::shared_ptr<RateHelper> > depoFutSwapInstruments;
309 depoFutSwapInstruments.push_back(d1w);
310 depoFutSwapInstruments.push_back(d1m);
311 depoFutSwapInstruments.push_back(fut1);
312 depoFutSwapInstruments.push_back(fut2);
313 depoFutSwapInstruments.push_back(fut3);
314 depoFutSwapInstruments.push_back(fut4);
315 depoFutSwapInstruments.push_back(fut5);
316 depoFutSwapInstruments.push_back(fut6);
317 depoFutSwapInstruments.push_back(fut7);
318 depoFutSwapInstruments.push_back(fut8);
319 depoFutSwapInstruments.push_back(s3y);
320 depoFutSwapInstruments.push_back(s5y);
321 depoFutSwapInstruments.push_back(s10y);
322 depoFutSwapInstruments.push_back(s15y);
323 boost::shared_ptr<YieldTermStructure> depoFutSwapTermStructure(new
324     PiecewiseFlatForward(settlementDate, depoFutSwapInstruments,
325         termStructureDayCounter, tolerance));
326
327
328 // A depo-FRA-swap curve
329 std::vector<boost::shared_ptr<RateHelper> > depoFRASwapInstruments;
330 depoFRASwapInstruments.push_back(d1w);
331 depoFRASwapInstruments.push_back(d1m);
332 depoFRASwapInstruments.push_back(d3m);
333 depoFRASwapInstruments.push_back(fra3x6);
334 depoFRASwapInstruments.push_back(fra6x9);
335 depoFRASwapInstruments.push_back(fra6x12);
336 depoFRASwapInstruments.push_back(s2y);
337 depoFRASwapInstruments.push_back(s3y);
338 depoFRASwapInstruments.push_back(s5y);
339 depoFRASwapInstruments.push_back(s10y);
340 depoFRASwapInstruments.push_back(s15y);
341 boost::shared_ptr<YieldTermStructure> depoFRASwapTermStructure(new
342     PiecewiseFlatForward(settlementDate, depoFRASwapInstruments,
343         termStructureDayCounter, tolerance));
344
345
346 // Term structures that will be used for pricing:
347 // the one used for discounting cash flows
348 Handle<YieldTermStructure> discountingTermStructure;

```

```

349 // the one used for forward rate forecasting
350 Handle<YieldTermStructure> forecastingTermStructure;
351
352
353 /*****
354  * SWAPS TO BE PRICED *
355  *****/
356
357 // constant nominal 1,000,000 Euro
358 Real nominal = 1000000.0;
359 // fixed leg
360 Frequency fixedLegFrequency = Annual;
361 BusinessDayConvention fixedLegConvention = Unadjusted;
362 BusinessDayConvention floatingLegConvention = ModifiedFollowing;
363 DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
364 Rate fixedRate = 0.04;
365
366 // floating leg
367 Frequency floatingLegFrequency = Semiannual;
368 boost::shared_ptr<Xibor> euriborIndex(new Euribor(6, Months,
369 forecastingTermStructure)); // using the forecasting curve
370 Spread spread = 0.0;
371
372 Integer lenghtInYears = 5;
373 bool payFixedRate = true;
374
375 Date maturity = calendar.advance(settlementDate, lenghtInYears, Years,
376 floatingLegConvention);
377 Schedule fixedSchedule(calendar, settlementDate, maturity,
378 fixedLegFrequency, fixedLegConvention);
379 Schedule floatSchedule(calendar, settlementDate, maturity,
380 floatingLegFrequency, floatingLegConvention);
381 SimpleSwap spot5YearSwap(
382 payFixedRate, nominal,
383 fixedSchedule, fixedRate, fixedLegDayCounter,
384 floatSchedule, euriborIndex, fixingDays, spread,
385 discountingTermStructure);
386
387 Date fwdStart = calendar.advance(settlementDate, 1, Years);
388 Date fwdMaturity = calendar.advance(fwdStart, lenghtInYears, Years,
389 floatingLegConvention);
390 Schedule fwdFixedSchedule(calendar, fwdStart, fwdMaturity,
391 fixedLegFrequency, fixedLegConvention);
392 Schedule fwdFloatSchedule(calendar, fwdStart, fwdMaturity,
393 floatingLegFrequency, floatingLegConvention);
394 SimpleSwap oneYearForward5YearSwap(
395 payFixedRate, nominal,
396 fwdFixedSchedule, fixedRate, fixedLegDayCounter,
397 fwdFloatSchedule, euriborIndex, fixingDays, spread,
398 discountingTermStructure);
399
400
401 /*****
402  * SWAP PRICING *
403  *****/
404
405 // utilities for reporting
406 std::vector<std::string> headers(4);
407 headers[0] = "term structure";
408 headers[1] = "net present value";
409 headers[2] = "fair spread";
410 headers[3] = "fair fixed rate";
411 std::string separator = " | ";
412 Size width = headers[0].size() + separator.size()
413 + headers[1].size() + separator.size()
414 + headers[2].size() + separator.size()
415 + headers[3].size() + separator.size() - 1;

```

```

416     std::string rule(width, '-'), dblrule(width, '=');
417     std::string tab(8, ' ');
418
419     // calculations
420
421     std::cout << dblrule << std::endl;
422     std::cout << "5-year market swap-rate = "
423         << std::setprecision(2) << io::rate(s5yRate->value())
424         << std::endl;
425     std::cout << dblrule << std::endl;
426
427     std::cout << tab << "5-years swap paying "
428         << io::rate(fixedRate) << std::endl;
429     std::cout << headers[0] << separator
430         << headers[1] << separator
431         << headers[2] << separator
432         << headers[3] << separator << std::endl;
433     std::cout << rule << std::endl;
434
435     Real NPV;
436     Rate fairRate;
437     Spread fairSpread;
438
439     // Of course, you're not forced to really use different curves
440     forecastingTermStructure.linkTo(depoSwapTermStructure);
441     discountingTermStructure.linkTo(depoSwapTermStructure);
442
443     NPV = spot5YearSwap.NPV();
444     fairSpread = spot5YearSwap.fairSpread();
445     fairRate = spot5YearSwap.fairRate();
446
447     std::cout << std::setw(headers[0].size())
448         << "depo-swap" << separator;
449     std::cout << std::setw(headers[1].size())
450         << std::fixed << std::setprecision(2) << NPV << separator;
451     std::cout << std::setw(headers[2].size())
452         << io::rate(fairSpread) << separator;
453     std::cout << std::setw(headers[3].size())
454         << io::rate(fairRate) << separator;
455     std::cout << std::endl;
456
457
458     // let's check that the 5 years swap has been correctly re-priced
459     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
460         "5-years swap mispriced by "
461         << io::rate(std::fabs(fairRate-s5yQuote)));
462
463
464     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
465     discountingTermStructure.linkTo(depoFutSwapTermStructure);
466
467     NPV = spot5YearSwap.NPV();
468     fairSpread = spot5YearSwap.fairSpread();
469     fairRate = spot5YearSwap.fairRate();
470
471     std::cout << std::setw(headers[0].size())
472         << "depo-fut-swap" << separator;
473     std::cout << std::setw(headers[1].size())
474         << std::fixed << std::setprecision(2) << NPV << separator;
475     std::cout << std::setw(headers[2].size())
476         << io::rate(fairSpread) << separator;
477     std::cout << std::setw(headers[3].size())
478         << io::rate(fairRate) << separator;
479     std::cout << std::endl;
480
481     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
482         "5-years swap mispriced!");

```

```

483
484
485     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
486     discountingTermStructure.linkTo(depoFRASwapTermStructure);
487
488     NPV = spot5YearSwap.NPV();
489     fairSpread = spot5YearSwap.fairSpread();
490     fairRate = spot5YearSwap.fairRate();
491
492     std::cout << std::setw(headers[0].size())
493               << "depo-FRA-swap" << separator;
494     std::cout << std::setw(headers[1].size())
495               << std::fixed << std::setprecision(2) << NPV << separator;
496     std::cout << std::setw(headers[2].size())
497               << io::rate(fairSpread) << separator;
498     std::cout << std::setw(headers[3].size())
499               << io::rate(fairRate) << separator;
500     std::cout << std::endl;
501
502     QL_REQUIRE(std::fabs(fairRate-s5yQuote)<1e-8,
503               "5-years swap mispriced!");
504
505
506     std::cout << rule << std::endl;
507
508     // now let's price the 1Y forward 5Y swap
509
510     std::cout << tab << "5-years, 1-year forward swap paying "
511               << io::rate(fixedRate) << std::endl;
512     std::cout << headers[0] << separator
513               << headers[1] << separator
514               << headers[2] << separator
515               << headers[3] << separator << std::endl;
516     std::cout << rule << std::endl;
517
518
519     forecastingTermStructure.linkTo(depoSwapTermStructure);
520     discountingTermStructure.linkTo(depoSwapTermStructure);
521
522     NPV = oneYearForward5YearSwap.NPV();
523     fairSpread = oneYearForward5YearSwap.fairSpread();
524     fairRate = oneYearForward5YearSwap.fairRate();
525
526     std::cout << std::setw(headers[0].size())
527               << "depo-swap" << separator;
528     std::cout << std::setw(headers[1].size())
529               << std::fixed << std::setprecision(2) << NPV << separator;
530     std::cout << std::setw(headers[2].size())
531               << io::rate(fairSpread) << separator;
532     std::cout << std::setw(headers[3].size())
533               << io::rate(fairRate) << separator;
534     std::cout << std::endl;
535
536
537     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
538     discountingTermStructure.linkTo(depoFutSwapTermStructure);
539
540     NPV = oneYearForward5YearSwap.NPV();
541     fairSpread = oneYearForward5YearSwap.fairSpread();
542     fairRate = oneYearForward5YearSwap.fairRate();
543
544     std::cout << std::setw(headers[0].size())
545               << "depo-fut-swap" << separator;
546     std::cout << std::setw(headers[1].size())
547               << std::fixed << std::setprecision(2) << NPV << separator;
548     std::cout << std::setw(headers[2].size())
549               << io::rate(fairSpread) << separator;

```

```

550     std::cout << std::setw(headers[3].size())
551               << io::rate(fairRate) << separator;
552     std::cout << std::endl;
553
554
555     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
556     discountingTermStructure.linkTo(depoFRASwapTermStructure);
557
558     NPV = oneYearForward5YearSwap.NPV();
559     fairSpread = oneYearForward5YearSwap.fairSpread();
560     fairRate = oneYearForward5YearSwap.fairRate();
561
562     std::cout << std::setw(headers[0].size())
563               << "depo-FRA-swap" << separator;
564     std::cout << std::setw(headers[1].size())
565               << std::fixed << std::setprecision(2) << NPV << separator;
566     std::cout << std::setw(headers[2].size())
567               << io::rate(fairSpread) << separator;
568     std::cout << std::setw(headers[3].size())
569               << io::rate(fairRate) << separator;
570     std::cout << std::endl;
571
572
573     // now let's say that the 5-years swap rate goes up to 4.60%.
574     // A smarter market element--say, connected to a data source-- would
575     // notice the change itself. Since we're using SimpleQuotes,
576     // we'll have to change the value manually--which forces us to
577     // downcast the handle and use the SimpleQuote
578     // interface. In any case, the point here is that a change in the
579     // value contained in the Quote triggers a new bootstrapping
580     // of the curve and a repricing of the swap.
581
582     boost::shared_ptr<SimpleQuote> fiveYearsRate =
583         boost::dynamic_pointer_cast<SimpleQuote>(s5yRate);
584     fiveYearsRate->setValue(0.0460);
585
586     std::cout << dblrule << std::endl;
587     std::cout << "5-year market swap-rate = "
588               << io::rate(s5yRate->value()) << std::endl;
589     std::cout << dblrule << std::endl;
590
591     std::cout << tab << "5-years swap paying "
592               << io::rate(fixedRate) << std::endl;
593     std::cout << headers[0] << separator
594               << headers[1] << separator
595               << headers[2] << separator
596               << headers[3] << separator << std::endl;
597     std::cout << rule << std::endl;
598
599     // now get the updated results
600     forecastingTermStructure.linkTo(depoSwapTermStructure);
601     discountingTermStructure.linkTo(depoSwapTermStructure);
602
603     NPV = spot5YearSwap.NPV();
604     fairSpread = spot5YearSwap.fairSpread();
605     fairRate = spot5YearSwap.fairRate();
606
607     std::cout << std::setw(headers[0].size())
608               << "depo-swap" << separator;
609     std::cout << std::setw(headers[1].size())
610               << std::fixed << std::setprecision(2) << NPV << separator;
611     std::cout << std::setw(headers[2].size())
612               << io::rate(fairSpread) << separator;
613     std::cout << std::setw(headers[3].size())
614               << io::rate(fairRate) << separator;
615     std::cout << std::endl;
616

```



```

617     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
618                "5-years swap mispriced!");
619
620
621     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
622     discountingTermStructure.linkTo(depoFutSwapTermStructure);
623
624     NPV = spot5YearSwap.NPV();
625     fairSpread = spot5YearSwap.fairSpread();
626     fairRate = spot5YearSwap.fairRate();
627
628     std::cout << std::setw(headers[0].size())
629               << "depo-fut-swap" << separator;
630     std::cout << std::setw(headers[1].size())
631               << std::fixed << std::setprecision(2) << NPV << separator;
632     std::cout << std::setw(headers[2].size())
633               << io::rate(fairSpread) << separator;
634     std::cout << std::setw(headers[3].size())
635               << io::rate(fairRate) << separator;
636     std::cout << std::endl;
637
638     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
639                "5-years swap mispriced!");
640
641
642     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
643     discountingTermStructure.linkTo(depoFRASwapTermStructure);
644
645     NPV = spot5YearSwap.NPV();
646     fairSpread = spot5YearSwap.fairSpread();
647     fairRate = spot5YearSwap.fairRate();
648
649     std::cout << std::setw(headers[0].size())
650               << "depo-FRA-swap" << separator;
651     std::cout << std::setw(headers[1].size())
652               << std::fixed << std::setprecision(2) << NPV << separator;
653     std::cout << std::setw(headers[2].size())
654               << io::rate(fairSpread) << separator;
655     std::cout << std::setw(headers[3].size())
656               << io::rate(fairRate) << separator;
657     std::cout << std::endl;
658
659     QL_REQUIRE(std::fabs(fairRate-s5yRate->value())<1e-8,
660                "5-years swap mispriced!");
661
662     std::cout << rule << std::endl;
663
664     // the 1Y forward 5Y swap changes as well
665
666     std::cout << tab << "5-years, 1-year forward swap paying "
667               << io::rate(fixedRate) << std::endl;
668     std::cout << headers[0] << separator
669               << headers[1] << separator
670               << headers[2] << separator
671               << headers[3] << separator << std::endl;
672     std::cout << rule << std::endl;
673
674
675     forecastingTermStructure.linkTo(depoSwapTermStructure);
676     discountingTermStructure.linkTo(depoSwapTermStructure);
677
678     NPV = oneYearForward5YearSwap.NPV();
679     fairSpread = oneYearForward5YearSwap.fairSpread();
680     fairRate = oneYearForward5YearSwap.fairRate();
681
682     std::cout << std::setw(headers[0].size())
683               << "depo-swap" << separator;

```

```

684         std::cout << std::setw(headers[1].size())
685             << std::fixed << std::setprecision(2) << NPV << separator;
686     std::cout << std::setw(headers[2].size())
687         << io::rate(fairSpread) << separator;
688     std::cout << std::setw(headers[3].size())
689         << io::rate(fairRate) << separator;
690     std::cout << std::endl;
691
692
693     forecastingTermStructure.linkTo(depoFutSwapTermStructure);
694     discountingTermStructure.linkTo(depoFutSwapTermStructure);
695
696     NPV = oneYearForward5YearSwap.NPV();
697     fairSpread = oneYearForward5YearSwap.fairSpread();
698     fairRate = oneYearForward5YearSwap.fairRate();
699
700     std::cout << std::setw(headers[0].size())
701         << "depo-fut-swap" << separator;
702     std::cout << std::setw(headers[1].size())
703         << std::fixed << std::setprecision(2) << NPV << separator;
704     std::cout << std::setw(headers[2].size())
705         << io::rate(fairSpread) << separator;
706     std::cout << std::setw(headers[3].size())
707         << io::rate(fairRate) << separator;
708     std::cout << std::endl;
709
710
711     forecastingTermStructure.linkTo(depoFRASwapTermStructure);
712     discountingTermStructure.linkTo(depoFRASwapTermStructure);
713
714     NPV = oneYearForward5YearSwap.NPV();
715     fairSpread = oneYearForward5YearSwap.fairSpread();
716     fairRate = oneYearForward5YearSwap.fairRate();
717
718     std::cout << std::setw(headers[0].size())
719         << "depo-FRA-swap" << separator;
720     std::cout << std::setw(headers[1].size())
721         << std::fixed << std::setprecision(2) << NPV << separator;
722     std::cout << std::setw(headers[2].size())
723         << io::rate(fairSpread) << separator;
724     std::cout << std::setw(headers[3].size())
725         << io::rate(fairRate) << separator;
726     std::cout << std::endl;
727
728     return 0;
729
730 } catch (std::exception& e) {
731     std::cout << e.what() << std::endl;
732     return 1;
733 } catch (...) {
734     std::cout << "unknown error" << std::endl;
735     return 1;
736 }
737 }
738

```

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

Test List

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 [Brent](#)

the correctness of the returned values is tested by checking them against known good results.

Class [BSMTermOperator](#)

coefficients are tested against constant BSM operator

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 [CapletLiborMarketModelProcess](#)

the correctness is tested by Monte-Carlo reproduction of caplet & ratchet npvs and comparison with Black pricing.

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 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.

Class 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.

Class 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.

Class FDEuropeanEngine

the correctness of the returned value is tested by checking it against analytic results.

Class FDShoutEngine

the correctness of the returned greeks is tested by reproducing numerical derivatives.

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.

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 [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 [RamdomizedLDS](#)

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 initializaion 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 [SimpleSwap](#)

- 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 [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 [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 [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.

Chapter 11

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)

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

Class [ShoutCondition](#)

unify the `intrinsicValues/Payoff` thing

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`

Warning:

This class assumes that the settlement date does not change between calls of `setTermStructure()`.

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 12

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 [FDDividendAmericanEngine](#)

method impliedVolatility() utterly fails

Class [G2](#)

This class was not tested enough to guarantee its functionality.

Class [HullWhite](#)

When the term structure is relinked, the r0 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 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 must still be checked. It is not guaranteed to yield the right results.

Chapter 13

Deprecated List

Class [MultiAsset](#)
use [MultiVariate](#) instead

Class [SingleAsset](#)
use [SingleVariate](#) instead

Member [YieldTermStructure::parRate](#)(Year tenor, Time t0, Frequency freq=[Annual](#), bool extrapolate=[false](#)) const
use the overload taking a vector of times

Member [IMMMonth](#)
use [IMM::Month](#) instead

Index

- ~Error
 - QuantLib::Error, 383
- accruedAmount
 - QuantLib::Bond, 234
- add
 - QuantLib::ExchangeRateManager, 395
 - QuantLib::GeneralStatistics, 478
 - QuantLib::IncrementalStatistics, 521
- addHoliday
 - QuantLib::Calendar, 255
- addSequence
 - QuantLib::IncrementalStatistics, 521
- adjust
 - QuantLib::Calendar, 255
- adjustValues
 - QuantLib::DiscretizedAsset, 357
- advance
 - QuantLib::Calendar, 256
- amount
 - QuantLib::CashFlow, 276
 - QuantLib::FixedRateCoupon, 427
 - QuantLib::IndexedCoupon, 524
 - QuantLib::ParCoupon, 705
 - QuantLib::Short, 762
 - QuantLib::SimpleCashFlow, 767
- Annual
 - datetime, 91
- apply
 - QuantLib::BlackScholesProcess, 218
 - QuantLib::CapletLiborMarketModel-
Process, 269
 - QuantLib::HestonProcess, 496
 - QuantLib::Merton76Process, 638
 - QuantLib::StochasticProcess, 796
 - QuantLib::StochasticProcess1D, 798
 - QuantLib::StochasticProcessArray, 802
- applyAfterApplying
 - QuantLib::BoundaryCondition, 235
 - QuantLib::DirichletBC, 347
 - QuantLib::NeumannBC, 660
- applyAfterSolving
 - QuantLib::BoundaryCondition, 236
 - QuantLib::DirichletBC, 347
 - QuantLib::NeumannBC, 661
- applyBeforeApplying
 - QuantLib::BoundaryCondition, 235
 - QuantLib::DirichletBC, 347
 - QuantLib::NeumannBC, 660
- applyBeforeSolving
 - QuantLib::BoundaryCondition, 235
 - QuantLib::DirichletBC, 347
 - QuantLib::NeumannBC, 660
- Asian option engines, 96
- AutomatedConversion
 - QuantLib::Money, 642
- averageShortfall
 - QuantLib::GenericRiskStatistics, 483
- BackwardFlatInterpolation
 - QuantLib::BackwardFlatInterpolation,
177
- Barrier option engines, 97
- BaseCurrencyConversion
 - QuantLib::Money, 642
- Basket option engines, 98
- BicubicSpline
 - QuantLib::BicubicSpline, 197
- BilinearInterpolation
 - QuantLib::BilinearInterpolation, 199
- Bimonthly
 - datetime, 91
- blackVarianceImpl
 - QuantLib::BlackVolatilityTerm-
Structure, 227
- BlackVarianceTermStructure
 - QuantLib::BlackVarianceTerm-
Structure, 225
- BlackVolatilityTermStructure
 - QuantLib::BlackVolatilityTerm-
Structure, 227
- blackVolImpl
 - QuantLib::BlackVarianceTerm-
Structure, 225
- BlackVolTermStructure
 - QuantLib::BlackVolTermStructure, 230
- BoundaryCondition
 - QuantLib::CubicSpline, 325
- BusinessDayConvention
 - datetime, 90

- calculate
 - QuantLib::Instrument, 528
 - QuantLib::LazyObject, 576
 - QuantLib::McSimulation, 633
- Calendar
 - QuantLib::Calendar, 255
- Calendars, 92
- calibrate
 - QuantLib::ShortRateModel, 765
- Cap/floor engines, 99
- CapletLiborMarketModelProcess
 - QuantLib::CapletLiborMarketModel-
Process, 269
- CapletVolatilityStructure
 - QuantLib::CapletVolatilityStructure,
271
- CapVolatilityStructure
 - QuantLib::CapVolatilityStructure, 273
- Ceiling
 - QuantLib::Rounding, 746
- cleanPrice
 - QuantLib::Bond, 233, 234
- Cliquet option engines, 100
- Closest
 - QuantLib::Rounding, 746
- compoundFactor
 - QuantLib::InterestRate, 532
- compoundForwardImpl
 - QuantLib::ExtendedDiscountCurve,
405
- ConversionType
 - QuantLib::Money, 642
- convexity
 - QuantLib::Cashflows, 278
- correlationMatrix
 - QuantLib::CovarianceDecomposition,
316
- Coupon
 - QuantLib::Coupon, 315
- covariance
 - QuantLib::EulerDiscretization, 387
 - QuantLib::StochasticProcess, 795
 - QuantLib::StochasticProcessArray, 802
- CovarianceDecomposition
 - QuantLib::CovarianceDecomposition,
316
- CubicSpline
 - QuantLib::CubicSpline, 325
- Currencies and FX rates, 85
- Currency
 - QuantLib::Currency, 333
- Date and time calculations, 89
- datetime
 - Annual, 91
 - Bimonthly, 91
 - BusinessDayConvention, 90
 - EveryFourthMonth, 91
 - Following, 91
 - Frequency, 91
 - IMMMonth, 91
 - ModifiedFollowing, 91
 - ModifiedPreceding, 91
 - MonthEndReference, 91
 - Monthly, 91
 - NoFrequency, 91
 - Once, 91
 - Preceding, 90
 - Quarterly, 91
 - Semiannual, 91
 - Unadjusted, 90
 - Weekday, 91
- Day counters, 94
- DayCounter
 - QuantLib::DayCounter, 341
- Debugging macros, 140
- debugMacros
 - QL_TRACE, 141
 - QL_TRACE_DISABLE, 140
 - QL_TRACE_ENABLE, 140
 - QL_TRACE_ENTER_FUNCTION, 141
 - QL_TRACE_EXIT_FUNCTION, 141
 - QL_TRACE_LOCATION, 142
 - QL_TRACE_ON, 141
 - QL_TRACE_VARIABLE, 142
- DEFINE_SEQUENCE_STAT_CONST_-
METHOD_DOUBLE
 - sequencestatistics.hpp, 1093
- DEFINE_SEQUENCE_STAT_CONST_-
METHOD_VOID
 - sequencestatistics.hpp, 1093
- Derived
 - QuantLib::ExchangeRate, 394
- Design patterns, 129
- diffusion
 - QuantLib::BlackScholesProcess, 218
 - QuantLib::EulerDiscretization, 386
- Direct
 - QuantLib::ExchangeRate, 394
- dirtyPrice
 - QuantLib::Bond, 233, 234
- discount
 - QuantLib::YieldTermStructure, 884
- DiscountCurve
 - yieldtermstructures, 131
- discountFactor
 - QuantLib::InterestRate, 532
- discountImpl

- QuantLib::CompoundForward, 297
- QuantLib::ForwardRateStructure, 441
- QuantLib::ZeroYieldStructure, 891
- Down
 - QuantLib::Rounding, 746
- downsideDeviation
 - QuantLib::GenericRiskStatistics, 482
 - QuantLib::IncrementalStatistics, 520
- downsideVariance
 - QuantLib::GenericRiskStatistics, 481
 - QuantLib::IncrementalStatistics, 520
- drift
 - QuantLib::BlackScholesProcess, 218
 - QuantLib::EulerDiscretization, 386
- duration
 - QuantLib::Cashflows, 278
- equivalentRate
 - QuantLib::InterestRate, 533
- Error
 - QuantLib::Error, 383
- errorEstimate
 - QuantLib::GeneralStatistics, 477
 - QuantLib::IncrementalStatistics, 520
- errors.hpp
 - QL_ASSERT, 968
 - QL_ENSURE, 968
 - QL_FAIL, 968
 - QL_REQUIRE, 968
- Eurex
 - QuantLib::Germany, 487
- evaluationDate
 - QuantLib::Settings, 759
- EveryFourthMonth
 - datetime, 91
- evolve
 - QuantLib::CapletLiborMarketModel-
Process, 269
 - QuantLib::StochasticProcess, 795
 - QuantLib::StochasticProcess1D, 798
- Exchange
 - QuantLib::Italy, 555
 - QuantLib::UnitedKingdom, 862
 - QuantLib::UnitedStates, 865
- ExchangeRate
 - QuantLib::ExchangeRate, 394
- expectation
 - QuantLib::OrnsteinUhlenbeckProcess, 699
 - QuantLib::StochasticProcess, 795
 - QuantLib::StochasticProcess1D, 798
 - QuantLib::StochasticProcessArray, 802
- expectationValue
 - QuantLib::GeneralStatistics, 478
- expectedShortfall
 - QuantLib::GenericRiskStatistics, 482
- Financial instruments, 115
- Finite-differences framework, 106
- FirstDerivative
 - QuantLib::CubicSpline, 325
- fixing
 - QuantLib::Index, 522
 - QuantLib::Xibor, 881
- Floor
 - QuantLib::Rounding, 746
- Following
 - datetime, 91
- format
 - QuantLib::Currency, 333
- formula
 - QuantLib::BlackModel, 214
- Forward option engines, 101
- ForwardFlatInterpolation
 - QuantLib::ForwardFlatInterpolation, 437
- forwardImpl
 - QuantLib::ZeroSpreadedTerm-
Structure, 889
- forwardRate
 - QuantLib::YieldTermStructure, 884
- FrankfurtStockExchange
 - QuantLib::Germany, 487
- freeze
 - QuantLib::LazyObject, 576
- Frequency
 - datetime, 91
- gaussianDownsideDeviation
 - QuantLib::GaussianStatistics, 466
- gaussianDownsideVariance
 - QuantLib::GaussianStatistics, 466
- gaussianExpectedShortfall
 - QuantLib::GaussianStatistics, 467
- gaussianPercentile
 - QuantLib::GaussianStatistics, 467
- gaussianPotentialUpside
 - QuantLib::GaussianStatistics, 467
- gaussianRegret
 - QuantLib::GaussianStatistics, 467
- gaussianTopPercentile
 - QuantLib::GaussianStatistics, 467
- gaussianValueAtRisk
 - QuantLib::GaussianStatistics, 467
- Generic macros, 135
- Handle
 - QuantLib::Handle, 491

- History
 - QuantLib::History, [498](#), [499](#)
- IMMMonth
 - datetime, [91](#)
- impliedRate
 - QuantLib::InterestRate, [533](#)
- impliedVolatility
 - QuantLib::OneAssetOption, [683](#)
 - QuantLib::SingleAssetOption, [779](#)
- irr
 - QuantLib::Cashflows, [278](#)
- isBusinessDay
 - QuantLib::Calendar, [255](#)
- isEndOfMonth
 - QuantLib::Calendar, [255](#)
- isHoliday
 - QuantLib::Calendar, [255](#)
- isOnTime
 - QuantLib::DiscretizedAsset, [357](#)
- Iterator support, [138](#)
- iteratorMacros
 - QL_FULL_ITERATOR_SUPPORT, [138](#)
- itmAssetProbability
 - QuantLib::BlackFormula, [210](#)
- itmCashProbability
 - QuantLib::BlackFormula, [210](#)
- itmProbability
 - QuantLib::BlackModel, [215](#)
- KnuthUniformRng
 - QuantLib::KnuthUniformRng, [566](#)
- kurtosis
 - QuantLib::GeneralStatistics, [477](#)
 - QuantLib::IncrementalStatistics, [521](#)
- Lagrange
 - QuantLib::CubicSpline, [325](#)
- lambda
 - QuantLib::CapletLiborMarketModel-
Process, [269](#)
- latestDate
 - QuantLib::DepositRateHelper, [345](#)
 - QuantLib::FixedCouponBondHelper, [425](#)
 - QuantLib::FraRateHelper, [448](#)
 - QuantLib::FuturesRateHelper, [450](#)
 - QuantLib::RateHelper, [740](#)
 - QuantLib::SwapRateHelper, [812](#)
- Lattice methods, [118](#)
- LecuyerUniformRng
 - QuantLib::LecuyerUniformRng, [579](#)
- limitMacros
 - QL_EPSILON, [136](#)
 - QL_MAX_INTEGER, [136](#)
 - QL_MAX_REAL, [136](#)
 - QL_MIN_INTEGER, [136](#)
 - QL_MIN_POSITIVE_REAL, [136](#)
 - QL_MIN_REAL, [136](#)
- LinearInterpolation
 - QuantLib::LinearInterpolation, [585](#)
- Link
 - QuantLib::Link, [588](#)
- linkTo
 - QuantLib::Handle, [491](#)
 - QuantLib::Link, [589](#)
- localVolImpl
 - QuantLib::LocalVolCurve, [592](#)
- LocalVolTermStructure
 - QuantLib::LocalVolTermStructure, [596](#)
- LogLinearInterpolation
 - QuantLib::LogLinearInterpolation, [598](#)
- lookup
 - QuantLib::ExchangeRateManager, [395](#)
- mandatoryTimes
 - QuantLib::DiscretizedAsset, [357](#)
 - QuantLib::DiscretizedDiscountBond, [359](#)
 - QuantLib::DiscretizedOption, [360](#)
- Market
 - QuantLib::Germany, [487](#)
 - QuantLib::Italy, [555](#)
 - QuantLib::UnitedKingdom, [862](#)
 - QuantLib::UnitedStates, [865](#)
- Math tools, [121](#)
- max
 - QuantLib::GeneralStatistics, [478](#)
 - QuantLib::IncrementalStatistics, [521](#)
- mean
 - QuantLib::GeneralStatistics, [477](#)
 - QuantLib::IncrementalStatistics, [520](#)
- MersenneTwisterUniformRng
 - QuantLib::MersenneTwisterUniform-
Rng, [636](#)
- min
 - QuantLib::GeneralStatistics, [477](#)
 - QuantLib::IncrementalStatistics, [521](#)
- miscMacros
 - QL_DUMMY_RETURN, [135](#)
 - QL_IO_INIT, [135](#)
- ModifiedFollowing
 - datetime, [91](#)
- ModifiedPreceding
 - datetime, [91](#)
- MonotonicCubicSpline
 - QuantLib::MonotonicCubicSpline, [643](#)
- Monte Carlo framework, [123](#)

- MonthEndReference
 - datetime, 91
- Monthly
 - datetime, 91
- name
 - QuantLib::Calendar, 255
 - QuantLib::DayCounter, 341
 - QuantLib::Index, 522
 - QuantLib::Xibor, 881
- NaturalCubicSpline
 - QuantLib::NaturalCubicSpline, 658
- NaturalMonotonicCubicSpline
 - QuantLib::NaturalMonotonicCubicSpline, 659
- next
 - QuantLib::KnuthUniformRng, 566
 - QuantLib::LecuyerUniformRng, 579
 - QuantLib::MersenneTwisterUniformRng, 636
- nextIMMdate
 - QuantLib::Date, 338
- nextRandomizer
 - QuantLib::RamdomizedLDS, 737
- nextWeekday
 - QuantLib::Date, 338
- NoConversion
 - QuantLib::Money, 642
- NoFrequency
 - datetime, 91
- None
 - QuantLib::Rounding, 746
- NotAKnot
 - QuantLib::CubicSpline, 325
- notifyObservers
 - QuantLib::Observable, 678
- npv
 - QuantLib::Cashflows, 277
- nthWeekday
 - QuantLib::Date, 338
- Numeric limits, 136
- Numeric types, 83
- Once
 - datetime, 91
- operator+=
 - QuantLib::Matrix, 608
- operator==
 - QuantLib::Calendar, 256
 - QuantLib::DayCounter, 341
- Output manipulators, 139
- parRate
 - QuantLib::YieldTermStructure, 884, 885
- partialRollback
 - QuantLib::Lattice, 571
 - QuantLib::NumericalMethod, 674
- percentile
 - QuantLib::GeneralStatistics, 478
- performCalculations
 - QuantLib::BarrierOption, 181
 - QuantLib::Bond, 234
 - QuantLib::ForwardVanillaOption, 446
 - QuantLib::Instrument, 528
 - QuantLib::LazyObject, 576
 - QuantLib::MultiAssetOption, 650
 - QuantLib::OneAssetOption, 684
 - QuantLib::OneAssetStrikedOption, 688
 - QuantLib::QuantoVanillaOption, 734
 - QuantLib::Stock, 803
 - QuantLib::Swap, 810
- Periodic
 - QuantLib::CubicSpline, 325
- postAdjustValues
 - QuantLib::DiscretizedAsset, 357
- postAdjustValuesImpl
 - QuantLib::DiscretizedAsset, 358
 - QuantLib::DiscretizedOption, 361
- potentialUpside
 - QuantLib::GenericRiskStatistics, 482
- preAdjustValues
 - QuantLib::DiscretizedAsset, 357
- preAdjustValuesImpl
 - QuantLib::DiscretizedAsset, 357
- Preceding
 - datetime, 90
- Pricing engines, 95
- pseudoSqrt
 - QuantLib::Matrix, 608
- ql/argsandresults.hpp, 895
- ql/calendar.hpp, 897
- ql/Calendars/beijing.hpp, 898
- ql/Calendars/bombay.hpp, 899
- ql/Calendars/bratislava.hpp, 900
- ql/Calendars/budapest.hpp, 901
- ql/Calendars/copenhagen.hpp, 902
- ql/Calendars/germany.hpp, 903
- ql/Calendars/helsinki.hpp, 904
- ql/Calendars/hongkong.hpp, 905
- ql/Calendars/istanbul.hpp, 906
- ql/Calendars/italy.hpp, 907
- ql/Calendars/johannesburg.hpp, 908
- ql/Calendars/jointcalendar.hpp, 909
- ql/Calendars/nullcalendar.hpp, 910
- ql/Calendars/oslo.hpp, 911
- ql/Calendars/prague.hpp, 912
- ql/Calendars/riyadh.hpp, 913

- ql/Calendars/seoul.hpp, 914
- ql/Calendars/singapore.hpp, 915
- ql/Calendars/stockholm.hpp, 916
- ql/Calendars/sydney.hpp, 917
- ql/Calendars/taipei.hpp, 918
- ql/Calendars/taiwan.hpp, 919
- ql/Calendars/target.hpp, 920
- ql/Calendars/tokyo.hpp, 921
- ql/Calendars/toronto.hpp, 922
- ql/Calendars/unitedkingdom.hpp, 923
- ql/Calendars/unitedstates.hpp, 924
- ql/Calendars/warsaw.hpp, 925
- ql/Calendars/wellington.hpp, 926
- ql/Calendars/zurich.hpp, 927
- ql/capvolstructures.hpp, 928
- ql/cashflow.hpp, 929
- ql/CashFlows/analysis.hpp, 930
- ql/CashFlows/basispointsensitivity.hpp, 931
- ql/CashFlows/cashflowvectors.hpp, 932
- ql/CashFlows/coupon.hpp, 933
- ql/CashFlows/fixedratecoupon.hpp, 934
- ql/CashFlows/floatingratecoupon.hpp, 935
- ql/CashFlows/inarrearindexedcoupon.hpp, 936
- ql/CashFlows/indexedcashflowvectors.hpp, 937
- ql/CashFlows/indexedcoupon.hpp, 938
- ql/CashFlows/parcoupon.hpp, 939
- ql/CashFlows/shortfloatingcoupon.hpp, 940
- ql/CashFlows/shortindexedcoupon.hpp, 941
- ql/CashFlows/simplecashflow.hpp, 942
- ql/CashFlows/timebasket.hpp, 943
- ql/CashFlows/upfrontindexedcoupon.hpp, 944
- ql/Currencies/africa.hpp, 945
- ql/Currencies/america.hpp, 946
- ql/Currencies/asia.hpp, 948
- ql/Currencies/europe.hpp, 950
- ql/Currencies/exchangeratemanager.hpp, 953
- ql/Currencies/oceania.hpp, 954
- ql/currency.hpp, 955
- ql/date.hpp, 956
- ql/daycounter.hpp, 959
- ql/DayCounters/actual360.hpp, 960
- ql/DayCounters/actual365fixed.hpp, 961
- ql/DayCounters/actualactual.hpp, 962
- ql/DayCounters/one.hpp, 963
- ql/DayCounters/simpledaycounter.hpp, 964
- ql/DayCounters/thirty360.hpp, 965
- ql/discretizedasset.hpp, 966
- ql/errors.hpp, 967
- ql/exchangerate.hpp, 970
- ql/exercise.hpp, 971
- ql/FiniteDifferences/americancondition.hpp, 972
- ql/FiniteDifferences/boundarycondition.hpp, 973
- ql/FiniteDifferences/bsmoperator.hpp, 974
- ql/FiniteDifferences/bsmtermoperator.hpp, 975
- ql/FiniteDifferences/cranknicolson.hpp, 976
- ql/FiniteDifferences/dminus.hpp, 977
- ql/FiniteDifferences/dplus.hpp, 978
- ql/FiniteDifferences/dplusdminus.hpp, 979
- ql/FiniteDifferences/dzero.hpp, 980
- ql/FiniteDifferences/expliciteuler.hpp, 981
- ql/FiniteDifferences/fdtypedefs.hpp, 982
- ql/FiniteDifferences/finitedifferencemodel.hpp, 983
- ql/FiniteDifferences/impliciteuler.hpp, 984
- ql/FiniteDifferences/mixedscheme.hpp, 985
- ql/FiniteDifferences/onefactoroperator.hpp, 986
- ql/FiniteDifferences/operatortraits.hpp, 987
- ql/FiniteDifferences/parallelevolver.hpp, 988
- ql/FiniteDifferences/shoutcondition.hpp, 989
- ql/FiniteDifferences/stepcondition.hpp, 990
- ql/FiniteDifferences/tridiagonaloperator.hpp, 991
- ql/FiniteDifferences/valueatcenter.hpp, 992
- ql/grid.hpp, 993
- ql/handle.hpp, 994
- ql/history.hpp, 995
- ql/index.hpp, 996
- ql/Indexes/audlibor.hpp, 997
- ql/Indexes/cadlibor.hpp, 998
- ql/Indexes/cdor.hpp, 999
- ql/Indexes/chflibor.hpp, 1000
- ql/Indexes/dkklibor.hpp, 1001
- ql/Indexes/euribor.hpp, 1002
- ql/Indexes/eurlibor.hpp, 1003
- ql/Indexes/gbplibor.hpp, 1004
- ql/Indexes/indexmanager.hpp, 1005
- ql/Indexes/jibor.hpp, 1006
- ql/Indexes/jpylibor.hpp, 1007
- ql/Indexes/libor.hpp, 1008
- ql/Indexes/nzdlbtor.hpp, 1009
- ql/Indexes/tibor.hpp, 1010
- ql/Indexes/trlibor.hpp, 1011
- ql/Indexes/usdlbtor.hpp, 1012
- ql/Indexes/xibor.hpp, 1013
- ql/Indexes/zibor.hpp, 1014
- ql/instrument.hpp, 1015
- ql/Instruments/asianoption.hpp, 1016
- ql/Instruments/barrieroption.hpp, 1017
- ql/Instruments/basketoption.hpp, 1018

- ql/Instruments/bond.hpp, 1019
- ql/Instruments/callabilityschedule.hpp, 1020
- ql/Instruments/capfloor.hpp, 1021
- ql/Instruments/cliquestoption.hpp, 1022
- ql/Instruments/convertiblebond.hpp, 1023
- ql/Instruments/dividendschedule.hpp, 1024
- ql/Instruments/dividendvanillaoption.hpp, 1025
- ql/Instruments/europeanoption.hpp, 1026
- ql/Instruments/fixedcouponbond.hpp, 1027
- ql/Instruments/floatingratebond.hpp, 1028
- ql/Instruments/forwardvanillaoption.hpp, 1029
- ql/Instruments/multiassetoption.hpp, 1030
- ql/Instruments/oneassetoption.hpp, 1031
- ql/Instruments/oneassetstrikedoption.hpp, 1032
- ql/Instruments/payoffs.hpp, 1033
- ql/Instruments/quantoforwardvanillaoption.hpp, 1034
- ql/Instruments/quantovanillaoption.hpp, 1035
- ql/Instruments/simpleswap.hpp, 1036
- ql/Instruments/stock.hpp, 1037
- ql/Instruments/swap.hpp, 1038
- ql/Instruments/swaption.hpp, 1039
- ql/Instruments/vanillaoption.hpp, 1040
- ql/Instruments/zerocouponbond.hpp, 1041
- ql/interestrate.hpp, 1042
- ql/Lattices/binomialtree.hpp, 1043
- ql/Lattices/bsmlattice.hpp, 1044
- ql/Lattices/lattice.hpp, 1045
- ql/Lattices/lattice1d.hpp, 1046
- ql/Lattices/lattice2d.hpp, 1047
- ql/Lattices/tree.hpp, 1048
- ql/Lattices/trinomialtree.hpp, 1049
- ql/Math/array.hpp, 1050
- ql/Math/backwardflatinterpolation.hpp, 1051
- ql/Math/beta.hpp, 1052
- ql/Math/bicubicsplineinterpolation.hpp, 1053
- ql/Math/bilinearinterpolation.hpp, 1054
- ql/Math/binomialdistribution.hpp, 1055
- ql/Math/bivariatenormaldistribution.hpp, 1056
- ql/Math/chisquaredistribution.hpp, 1057
- ql/Math/choleskydecomposition.hpp, 1058
- ql/Math/comparison.hpp, 1059
- ql/Math/convergencestatistics.hpp, 1060
- ql/Math/cubicspline.hpp, 1061
- ql/Math/discrepancystatistics.hpp, 1062
- ql/Math/errorfunction.hpp, 1063
- ql/Math/extrapolation.hpp, 1064
- ql/Math/factorial.hpp, 1065
- ql/Math/forwardflatinterpolation.hpp, 1066
- ql/Math/functional.hpp, 1067
- ql/Math/gammadistribution.hpp, 1068
- ql/Math/gaussianorthogonalpolynomial.hpp, 1069
- ql/Math/gaussianquadratures.hpp, 1070
- ql/Math/gaussianstatistics.hpp, 1072
- ql/Math/generalstatistics.hpp, 1073
- ql/Math/incompletegamma.hpp, 1074
- ql/Math/incrementalstatistics.hpp, 1075
- ql/Math/interpolation.hpp, 1076
- ql/Math/interpolation2D.hpp, 1077
- ql/Math/kronrodintegral.hpp, 1078
- ql/Math/lexicographicalview.hpp, 1079
- ql/Math/linearinterpolation.hpp, 1080
- ql/Math/loglinearinterpolation.hpp, 1081
- ql/Math/matrix.hpp, 1082
- ql/Math/multicubicspline.hpp, 1083
- ql/Math/normaldistribution.hpp, 1085
- ql/Math/poissondistribution.hpp, 1086
- ql/Math/primenumbers.hpp, 1087
- ql/Math/pseudosqrt.hpp, 1088
- ql/Math/riskstatistics.hpp, 1089
- ql/Math/rounding.hpp, 1090
- ql/Math/sampledcurve.hpp, 1091
- ql/Math/segmentintegral.hpp, 1092
- ql/Math/sequencestatistics.hpp, 1093
- ql/Math/simpsonintegral.hpp, 1095
- ql/Math/statistics.hpp, 1096
- ql/Math/svd.hpp, 1097
- ql/Math/symmetriceigenvalues.hpp, 1098
- ql/Math/symmetricschurdecomposition.hpp, 1099
- ql/Math/tkreigendecomposition.hpp, 1100
- ql/Math/trapezoidintegral.hpp, 1101
- ql/money.hpp, 1102
- ql/MonteCarlo/brownianbridge.hpp, 1103
- ql/MonteCarlo/getcovariance.hpp, 1104
- ql/MonteCarlo/mctrails.hpp, 1105
- ql/MonteCarlo/mctypedefs.hpp, 1106
- ql/MonteCarlo/montecarlomodel.hpp, 1107
- ql/MonteCarlo/multipath.hpp, 1108
- ql/MonteCarlo/multipathgenerator.hpp, 1109
- ql/MonteCarlo/path.hpp, 1110
- ql/MonteCarlo/pathgenerator.hpp, 1111
- ql/MonteCarlo/pathpricer.hpp, 1112
- ql/MonteCarlo/sample.hpp, 1113
- ql/numericalmethod.hpp, 1114
- ql/Optimization/armijo.hpp, 1115
- ql/Optimization/conjugategradient.hpp, 1116
- ql/Optimization/constraint.hpp, 1117

- ql/Optimization/costfunction.hpp, 1118
- ql/Optimization/criteria.hpp, 1119
- ql/Optimization/leastsquare.hpp, 1120
- ql/Optimization/linearch.hpp, 1121
- ql/Optimization/method.hpp, 1122
- ql/Optimization/problem.hpp, 1123
- ql/Optimization/simplex.hpp, 1124
- ql/Optimization/steepestdescent.hpp, 1125
- ql/option.hpp, 1126
- ql/Patterns/bridge.hpp, 1127
- ql/Patterns/composite.hpp, 1128
- ql/Patterns/curiouslyrecurring.hpp, 1129
- ql/Patterns/lazyobject.hpp, 1130
- ql/Patterns/observable.hpp, 1131
- ql/Patterns/singleton.hpp, 1132
- ql/Patterns/visitor.hpp, 1133
- ql/payoff.hpp, 1134
- ql/Pricers/discretegeometricaso.hpp, 1135
- ql/Pricers/mccliquetoption.hpp, 1136
- ql/Pricers/mcdiscretearithmeticaso.hpp, 1137
- ql/Pricers/mceverest.hpp, 1138
- ql/Pricers/mchimalaya.hpp, 1139
- ql/Pricers/mcmaxbasket.hpp, 1140
- ql/Pricers/mcpagoda.hpp, 1141
- ql/Pricers/mcperformanceoption.hpp, 1142
- ql/Pricers/mcpricer.hpp, 1143
- ql/Pricers/singleassetoption.hpp, 1144
- ql/pricingengine.hpp, 1145
- ql/PricingEngines/americanpayoffatexpiry.hpp, 1146
- ql/PricingEngines/americanpayoffathit.hpp, 1147
- ql/PricingEngines/Asian/analytic_cont_-geom_av_price.hpp, 1148
- ql/PricingEngines/Asian/analytic_discr_-geom_av_price.hpp, 1149
- ql/PricingEngines/Asian/mc_discr_arith_-av_price.hpp, 1150
- ql/PricingEngines/Asian/mc_discr_geom_-av_price.hpp, 1151
- ql/PricingEngines/Asian/mcdiscreteasianengine.hpp, 1152
- ql/PricingEngines/Barrier/analyticbarrierengine.hpp, 1153
- ql/PricingEngines/Barrier/mcbarrierengine.hpp, 1154
- ql/PricingEngines/Basket/mcamericanbasketengine.hpp, 1155
- ql/PricingEngines/Basket/mcbasketengine.hpp, 1156
- ql/PricingEngines/Basket/stulzengine.hpp, 1157
- ql/PricingEngines/blackformula.hpp, 1158
- ql/PricingEngines/blackmodel.hpp, 1159
- ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp, 1160
- ql/PricingEngines/CapFloor/blackcapfloorengine.hpp, 1161
- ql/PricingEngines/CapFloor/discretizedcapfloor.hpp, 1162
- ql/PricingEngines/CapFloor/treecapfloorengine.hpp, 1163
- ql/PricingEngines/Cliquet/analyticcliquetengine.hpp, 1164
- ql/PricingEngines/Cliquet/analyticperformanceengine.hpp, 1165
- ql/PricingEngines/Cliquet/mccliquetengine.hpp, 1166
- ql/PricingEngines/Forward/forwardengine.hpp, 1167
- ql/PricingEngines/Forward/forwardperformanceengine.hpp, 1168
- ql/PricingEngines/genericmodelengine.hpp, 1169
- ql/PricingEngines/greeks.hpp, 1170
- ql/PricingEngines/latticeshortratemodelengine.hpp, 1171
- ql/PricingEngines/mcsimulation.hpp, 1172
- ql/PricingEngines/Quanto/quantoengine.hpp, 1173
- ql/PricingEngines/Swaption/blackswaptionengine.hpp, 1174
- ql/PricingEngines/Swaption/discretizedswaption.hpp, 1175
- ql/PricingEngines/Swaption/g2swaptionengine.hpp, 1176
- ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp, 1177
- ql/PricingEngines/Swaption/treeswaptionengine.hpp, 1178
- ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp, 1179
- ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp, 1180
- ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp, 1181
- ql/PricingEngines/Vanilla/analytichestonengine.hpp, 1182
- ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp, 1183
- ql/PricingEngines/Vanilla/batesengine.hpp, 1184
- ql/PricingEngines/Vanilla/binomialengine.hpp, 1185
- ql/PricingEngines/Vanilla/bjerkstundstenslandengine.hpp, 1186

- ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp, 1219
- ql/PricingEngines/Vanilla/fdamericanengine.hpp, 1188
- ql/PricingEngines/Vanilla/fdbermudanengine.hpp, 1189
- ql/PricingEngines/Vanilla/fddividendamercianengine.hpp, 1190
- ql/PricingEngines/Vanilla/fddividendengine.hpp, 1191
- ql/PricingEngines/Vanilla/fddividendeuropeanengine.hpp, 1192
- ql/PricingEngines/Vanilla/fddividendshoutengine.hpp, 1193
- ql/PricingEngines/Vanilla/fdeuropeanengine.hpp, 1194
- ql/PricingEngines/Vanilla/fdmultiperiodengine.hpp, 1195
- ql/PricingEngines/Vanilla/fdshoutengine.hpp, 1196
- ql/PricingEngines/Vanilla/fdstepconditionengine.hpp, 1197
- ql/PricingEngines/Vanilla/fdvanillaengine.hpp, 1198
- ql/PricingEngines/Vanilla/integralengine.hpp, 1199
- ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp, 1200
- ql/PricingEngines/Vanilla/juquadraticengine.hpp, 1201
- ql/PricingEngines/Vanilla/mcdigitalengine.hpp, 1202
- ql/PricingEngines/Vanilla/mceuropeanengine.hpp, 1203
- ql/PricingEngines/Vanilla/mceuropeanhestonengine.hpp, 1204
- ql/PricingEngines/Vanilla/mchestonengine.hpp, 1205
- ql/PricingEngines/Vanilla/mcvanillaengine.hpp, 1206
- ql/Processes/blackscholesprocess.hpp, 1207
- ql/Processes/capletlmmprocess.hpp, 1208
- ql/Processes/defaulttable.hpp, 1209
- ql/Processes/eulerdiscretization.hpp, 1210
- ql/Processes/geometricbrownianprocess.hpp, 1211
- ql/Processes/hestonprocess.hpp, 1212
- ql/Processes/merton76process.hpp, 1213
- ql/Processes/ornsteinuhlenbeckprocess.hpp, 1214
- ql/Processes/squarerootprocess.hpp, 1215
- ql/Processes/stochasticprocessarray.hpp, 1216
- ql/qldefines.hpp, 1217
- ql/quote.hpp, 1219
- ql/RandomNumbers/boxmullergaussianrng.hpp, 1220
- ql/RandomNumbers/centrallimitgaussianrng.hpp, 1221
- ql/RandomNumbers/faurersg.hpp, 1222
- ql/RandomNumbers/haltonrsg.hpp, 1223
- ql/RandomNumbers/inversecumulativerng.hpp, 1224
- ql/RandomNumbers/inversecumulativersg.hpp, 1225
- ql/RandomNumbers/knuthuniformrng.hpp, 1226
- ql/RandomNumbers/lecuyeruniformrng.hpp, 1227
- ql/RandomNumbers/mt19937uniformrng.hpp, 1228
- ql/RandomNumbers/randomizedlds.hpp, 1229
- ql/RandomNumbers/randomsequencegenerator.hpp, 1230
- ql/RandomNumbers/rngtraits.hpp, 1231
- ql/RandomNumbers/seedgenerator.hpp, 1233
- ql/RandomNumbers/sobolrsg.hpp, 1234
- ql/schedule.hpp, 1235
- ql/settings.hpp, 1236
- ql/ShortRateModels/calibrationhelper.hpp, 1237
- ql/ShortRateModels/CalibrationHelpers/caphelper.hpp, 1238
- ql/ShortRateModels/CalibrationHelpers/hestonmodelhelper.hpp, 1239
- ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp, 1240
- ql/ShortRateModels/model.hpp, 1241
- ql/ShortRateModels/onefactormodel.hpp, 1242
- ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp, 1243
- ql/ShortRateModels/OneFactorModels/coxingersollross.hpp, 1244
- ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp, 1245
- ql/ShortRateModels/OneFactorModels/hullwhite.hpp, 1246
- ql/ShortRateModels/OneFactorModels/vasicek.hpp, 1247
- ql/ShortRateModels/parameter.hpp, 1248
- ql/ShortRateModels/twofactormodel.hpp, 1249
- ql/ShortRateModels/TwoFactorModels/batesmodel.hpp, 1250

- ql/ShortRateModels/TwoFactorModels/g2.hpp, 1251
- ql/ShortRateModels/TwoFactorModels/hestonmodel.hpp, 1252
- ql/solver1d.hpp, 1253
- ql/Solvers1D/bisection.hpp, 1254
- ql/Solvers1D/brent.hpp, 1255
- ql/Solvers1D/falseposition.hpp, 1256
- ql/Solvers1D/newton.hpp, 1257
- ql/Solvers1D/newtonsafe.hpp, 1258
- ql/Solvers1D/ridder.hpp, 1259
- ql/Solvers1D/secant.hpp, 1260
- ql/stochasticprocess.hpp, 1261
- ql/swaptionvolstructure.hpp, 1262
- ql/termstructure.hpp, 1263
- ql/TermStructures/affinetermstructure.hpp, 1264
- ql/TermStructures/bondhelpers.hpp, 1265
- ql/TermStructures/bootstraptraits.hpp, 1266
- ql/TermStructures/compoundforward.hpp, 1267
- ql/TermStructures/discountcurve.hpp, 1268
- ql/TermStructures/drifttermstructure.hpp, 1269
- ql/TermStructures/extendeddiscountcurve.hpp, 1270
- ql/TermStructures/flatforward.hpp, 1271
- ql/TermStructures/forwardcurve.hpp, 1272
- ql/TermStructures/forwardspreadetermstructure.hpp, 1273
- ql/TermStructures/forwardstructure.hpp, 1274
- ql/TermStructures/implicittermstructure.hpp, 1275
- ql/TermStructures/piecewiseflatforward.hpp, 1276
- ql/TermStructures/piecewiseyieldcurve.hpp, 1277
- ql/TermStructures/quantotermstructure.hpp, 1278
- ql/TermStructures/ratehelpers.hpp, 1279
- ql/TermStructures/zerocurve.hpp, 1280
- ql/TermStructures/zerospreadetermstructure.hpp, 1281
- ql/TermStructures/zeroyieldstructure.hpp, 1282
- ql/timegrid.hpp, 1283
- ql/types.hpp, 1284
- ql/Utilities/dataformatters.hpp, 1286
- ql/Utilities/dataparsers.hpp, 1287
- ql/Utilities/disposable.hpp, 1288
- ql/Utilities/null.hpp, 1289
- ql/Utilities/observablevalue.hpp, 1290
- ql/Utilities/steppingiterator.hpp, 1291
- ql/Utilities/strings.hpp, 1292
- ql/Utilities/tracing.hpp, 1293
- ql/Volatilities/blackconstantvol.hpp, 1295
- ql/Volatilities/blackvariancecurve.hpp, 1296
- ql/Volatilities/blackvariancesurface.hpp, 1297
- ql/Volatilities/capflatvolvector.hpp, 1298
- ql/Volatilities/capletconstantvol.hpp, 1299
- ql/Volatilities/capletvariancecurve.hpp, 1300
- ql/Volatilities/implicitvoltermstructure.hpp, 1301
- ql/Volatilities/localconstantvol.hpp, 1302
- ql/Volatilities/localvolcurve.hpp, 1303
- ql/Volatilities/localvolsurface.hpp, 1304
- ql/Volatilities/swaptionvolmatrix.hpp, 1305
- ql/voltermstructure.hpp, 1306
- ql/yieldtermstructure.hpp, 1307
- QL_ASSERT
 - errors.hpp, 968
- QL_DUMMY_RETURN
 - miscMacros, 135
- QL_ENSURE
 - errors.hpp, 968
- QL_EPSILON
 - limitMacros, 136
- QL_FAIL
 - errors.hpp, 968
- QL_FULL_ITERATOR_SUPPORT
 - iteratorMacros, 138
- QL_IO_INIT
 - miscMacros, 135
- QL_MAX_INTEGER
 - limitMacros, 136
- QL_MAX_REAL
 - limitMacros, 136
- QL_MIN_INTEGER
 - limitMacros, 136
- QL_MIN_POSITIVE_REAL
 - limitMacros, 136
- QL_MIN_REAL
 - limitMacros, 136
- QL_REQUIRE
 - errors.hpp, 968
- QL_TRACE
 - debugMacros, 141
- QL_TRACE_DISABLE
 - debugMacros, 140
- QL_TRACE_ENABLE
 - debugMacros, 140
- QL_TRACE_ENTER_FUNCTION
 - debugMacros, 141
- QL_TRACE_EXIT_FUNCTION
 - debugMacros, 141
- QL_TRACE_LOCATION

- debugMacros, 142
- QL_TRACE_ON
 - debugMacros, 141
- QL_TRACE_VARIABLE
 - debugMacros, 142
- QL_TYPENAME
 - templateMacros, 137
- QuantLib macros, 134
- QuantLib::Actual360, 143
- QuantLib::Actual365Fixed, 144
- QuantLib::ActualActual, 145
- QuantLib::AcyclicVisitor, 146
- QuantLib::AdditiveEQPBinomialTree, 147
- QuantLib::AffineModel, 148
- QuantLib::AffineTermStructure, 149
- QuantLib::AffineTermStructure
 - update, 150
- QuantLib::AmericanCondition, 151
- QuantLib::AmericanExercise, 152
- QuantLib::AmericanPayoffAtExpiry, 153
- QuantLib::AmericanPayoffAtHit, 154
- QuantLib::AnalyticBarrierEngine, 155
- QuantLib::AnalyticCapFloorEngine, 156
- QuantLib::AnalyticCliquetEngine, 157
- QuantLib::AnalyticContinuousGeometricAveragePriceAsianEngine, 158
- QuantLib::AnalyticDigitalAmericanEngine, 159
- QuantLib::AnalyticDiscreteGeometricAveragePriceAsianEngine, 160
- QuantLib::AnalyticDividendEuropeanEngine, 161
- QuantLib::AnalyticEuropeanEngine, 162
- QuantLib::AnalyticHestonEngine, 163
- QuantLib::AnalyticPerformanceEngine, 164
- QuantLib::Arguments, 165
- QuantLib::ArmijoLineSearch, 166
- QuantLib::Array, 167
- QuantLib::ARSCurrency, 170
- QuantLib::AssetOrNothingPayoff, 171
- QuantLib::ATSCurrency, 172
- QuantLib::AUDCurrency, 173
- QuantLib::AUDLibor, 174
- QuantLib::Average, 175
- QuantLib::BackwardFlat, 176
- QuantLib::BackwardFlatInterpolation, 177
- QuantLib::BackwardFlatInterpolation
 - BackwardFlatInterpolation, 177
- QuantLib::BaroneAdesiWhaleyApproximationEngine, 178
- QuantLib::Barrier, 179
- QuantLib::BarrierOption, 180
- QuantLib::BarrierOption
 - performCalculations, 181
 - setupArguments, 181
- QuantLib::BarrierOption::arguments, 182
- QuantLib::BarrierOption::engine, 183
- QuantLib::BasketOption, 184
- QuantLib::BasketOption
 - setupArguments, 185
- QuantLib::BasketOption::arguments, 186
- QuantLib::BasketOption::engine, 187
- QuantLib::BatesEngine, 188
- QuantLib::BatesModel, 190
- QuantLib::BDTCurrency, 191
- QuantLib::BEFCurrency, 192
- QuantLib::Beijing, 193
- QuantLib::BermudanExercise, 194
- QuantLib::BGLCurrency, 195
- QuantLib::Bicubic, 196
- QuantLib::BicubicSpline, 197
- QuantLib::BicubicSpline
 - BicubicSpline, 197
- QuantLib::Bilinear, 198
- QuantLib::BilinearInterpolation, 199
- QuantLib::BilinearInterpolation
 - BilinearInterpolation, 199
- QuantLib::BinomialDistribution, 200
- QuantLib::BinomialTree, 201
- QuantLib::BinomialVanillaEngine, 202
- QuantLib::Bisection, 203
- QuantLib::BivariateCumulativeNormalDistributionDr78, 204
- QuantLib::BivariateCumulativeNormalDistributionWe04DP, 205
- QuantLib::Bjerk Sund Stensland Approximation Engine, 206
- QuantLib::BlackCapFloorEngine, 207
- QuantLib::BlackConstantVol, 208
- QuantLib::BlackFormula, 210
- QuantLib::BlackFormula
 - itmAssetProbability, 210
 - itmCashProbability, 210
- QuantLib::BlackKarasinski, 212
- QuantLib::BlackKarasinski::Dynamics, 213
- QuantLib::BlackModel, 214
- QuantLib::BlackModel
 - formula, 214
 - itmProbability, 215
 - update, 214
- QuantLib::BlackScholesLattice, 216
- QuantLib::BlackScholesProcess, 217
- QuantLib::BlackScholesProcess
 - apply, 218
 - diffusion, 218
 - drift, 218
 - time, 218
 - update, 218

- QuantLib::BlackSwaptionEngine, 219
- QuantLib::BlackVarianceCurve, 220
- QuantLib::BlackVarianceSurface, 222
- QuantLib::BlackVarianceTermStructure, 224
- QuantLib::BlackVarianceTermStructure
 - BlackVarianceTermStructure, 225
 - blackVolImpl, 225
- QuantLib::BlackVolatilityTermStructure, 226
- QuantLib::BlackVolatilityTermStructure
 - blackVarianceImpl, 227
 - BlackVolatilityTermStructure, 227
- QuantLib::BlackVolTermStructure, 228
- QuantLib::BlackVolTermStructure
 - BlackVolTermStructure, 230
- QuantLib::Bombay, 231
- QuantLib::Bond, 232
- QuantLib::Bond
 - accruedAmount, 234
 - cleanPrice, 233, 234
 - dirtyPrice, 233, 234
 - performCalculations, 234
 - yield, 234
- QuantLib::BoundaryCondition, 235
- QuantLib::BoundaryCondition
 - applyAfterApplying, 235
 - applyAfterSolving, 236
 - applyBeforeApplying, 235
 - applyBeforeSolving, 235
 - setTime, 236
 - Side, 235
- QuantLib::BoundaryConstraint, 237
- QuantLib::BoxMullerGaussianRng, 238
- QuantLib::BPSBasketCalculator, 239
- QuantLib::BPSCalculator, 240
- QuantLib::Bratislava, 241
- QuantLib::Brent, 242
- QuantLib::Bridge, 243
- QuantLib::BRLCurrency, 244
- QuantLib::BrownianBridge, 245
- QuantLib::BSMOperator, 246
- QuantLib::BSMTermOperator, 247
- QuantLib::Budapest, 248
- QuantLib::BYRCurrency, 249
- QuantLib::CADCurrency, 250
- QuantLib::CADLibor, 251
- QuantLib::Calendar, 252
- QuantLib::Calendar
 - addHoliday, 255
 - adjust, 255
 - advance, 256
 - Calendar, 255
 - isBusinessDay, 255
 - isEndOfMonth, 255
 - isHoliday, 255
 - name, 255
 - operator==, 256
 - removeHoliday, 255
- QuantLib::Calendar::WesternImpl, 257
- QuantLib::CalendarImpl, 258
- QuantLib::CalibrationHelper, 259
- QuantLib::CalibrationHelper
 - update, 260
- QuantLib::Cap, 261
- QuantLib::CapFloor, 262
- QuantLib::CapFloor
 - setupArguments, 263
- QuantLib::CapFloor::arguments, 264
- QuantLib::CapFloor::results, 265
- QuantLib::CapletConstantVolatility, 266
- QuantLib::CapletLiborMarketModelProcess, 268
- QuantLib::CapletLiborMarketModelProcess
 - apply, 269
 - CapletLiborMarketModelProcess, 269
 - evolve, 269
 - lambda, 269
- QuantLib::CapletVolatilityStructure, 270
- QuantLib::CapletVolatilityStructure
 - CapletVolatilityStructure, 271
- QuantLib::CapVolatilityStructure, 272
- QuantLib::CapVolatilityStructure
 - CapVolatilityStructure, 273
- QuantLib::CapVolatilityVector, 274
- QuantLib::CapVolatilityVector
 - update, 275
- QuantLib::CashFlow, 276
- QuantLib::CashFlow
 - amount, 276
- QuantLib::Cashflows, 277
- QuantLib::Cashflows
 - convexity, 278
 - duration, 278
 - irr, 278
 - npv, 277
- QuantLib::CashOrNothingPayoff, 279
- QuantLib::Cdor, 280
- QuantLib::CeilingTruncation, 281
- QuantLib::CHFCurrency, 282
- QuantLib::CHFLibor, 283
- QuantLib::CLGaussianRng, 284
- QuantLib::CliquetOption, 285
- QuantLib::CliquetOption
 - setupArguments, 286
- QuantLib::CliquetOption::arguments, 287
- QuantLib::CliquetOption::engine, 288
- QuantLib::ClosestRounding, 289
- QuantLib::CLPCurrency, 290
- QuantLib::CNYCurrency, 291

- QuantLib::Collar, [292](#)
- QuantLib::Composite, [293](#)
- QuantLib::CompositeConstraint, [294](#)
- QuantLib::CompositeQuote, [295](#)
- QuantLib::CompositeQuote
 - update, [295](#)
- QuantLib::CompoundForward, [296](#)
- QuantLib::CompoundForward
 - discountImpl, [297](#)
 - zeroYieldImpl, [297](#)
- QuantLib::ConjugateGradient, [298](#)
- QuantLib::ConstantParameter, [299](#)
- QuantLib::Constraint, [300](#)
- QuantLib::ConstraintImpl, [301](#)
- QuantLib::ContinuousAveragingAsianOption, [302](#)
- QuantLib::ContinuousAveragingAsian-Option
 - setupArguments, [303](#)
- QuantLib::ContinuousAveragingAsianOption::arguments, [304](#)
- QuantLib::ContinuousAveragingAsianOption::engine, [305](#)
- QuantLib::ConvergenceStatistics, [306](#)
- QuantLib::ConvertibleBond::option, [307](#)
- QuantLib::ConvertibleBond::option
 - setupArguments, [307](#)
- QuantLib::ConvertibleBond::option::arguments, [309](#)
- QuantLib::ConvertibleBond::option::engine, [310](#)
- QuantLib::COPCurrency, [311](#)
- QuantLib::Copenhagen, [312](#)
- QuantLib::CostFunction, [313](#)
- QuantLib::Coupon, [314](#)
- QuantLib::Coupon
 - Coupon, [315](#)
- QuantLib::CovarianceDecomposition, [316](#)
- QuantLib::CovarianceDecomposition
 - correlationMatrix, [316](#)
 - CovarianceDecomposition, [316](#)
 - standardDeviations, [316](#)
 - variances, [316](#)
- QuantLib::CoxIngersollRoss, [317](#)
- QuantLib::CoxIngersollRoss::Dynamics, [319](#)
- QuantLib::CoxRossRubinstein, [320](#)
- QuantLib::CrankNicolson, [321](#)
- QuantLib::Cubic, [323](#)
- QuantLib::CubicSpline, [324](#)
 - FirstDerivative, [325](#)
 - Lagrange, [325](#)
 - NotAKnot, [325](#)
 - Periodic, [325](#)
 - SecondDerivative, [325](#)
- QuantLib::CubicSpline
 - BoundaryCondition, [325](#)
 - CubicSpline, [325](#)
- QuantLib::CumulativeBinomialDistribution, [326](#)
- QuantLib::CumulativeNormalDistribution, [327](#)
- QuantLib::CumulativePoissonDistribution, [328](#)
- QuantLib::CuriouslyRecurringTemplate, [329](#)
- QuantLib::Currency, [330](#)
- QuantLib::Currency
 - Currency, [333](#)
 - format, [333](#)
- QuantLib::CYPCurrency, [334](#)
- QuantLib::CZKCurrency, [335](#)
- QuantLib::Date, [336](#)
- QuantLib::Date
 - nextIMMdate, [338](#)
 - nextWeekday, [338](#)
 - nthWeekday, [338](#)
- QuantLib::DayCounter, [340](#)
- QuantLib::DayCounter
 - DayCounter, [341](#)
 - name, [341](#)
 - operator==, [341](#)
- QuantLib::DayCounterImpl, [342](#)
- QuantLib::DEMCurrency, [343](#)
- QuantLib::DepositRateHelper, [344](#)
- QuantLib::DepositRateHelper
 - latestDate, [345](#)
 - setTermStructure, [344](#)
- QuantLib::DerivedQuote, [346](#)
- QuantLib::DerivedQuote
 - update, [346](#)
- QuantLib::DirichletBC, [347](#)
- QuantLib::DirichletBC
 - applyAfterApplying, [347](#)
 - applyAfterSolving, [347](#)
 - applyBeforeApplying, [347](#)
 - applyBeforeSolving, [347](#)
 - setTime, [348](#)
- QuantLib::Discount, [349](#)
- QuantLib::DiscrepancyStatistics, [350](#)
- QuantLib::DiscreteAveragingAsianOption, [351](#)
- QuantLib::DiscreteAveragingAsianOption
 - setupArguments, [352](#)
- QuantLib::DiscreteAveragingAsianOption::arguments, [353](#)
- QuantLib::DiscreteAveragingAsianOption::engine, [354](#)
- QuantLib::DiscreteGeometricASO, [355](#)

- QuantLib::DiscretizedAsset, 356
- QuantLib::DiscretizedAsset
 - adjustValues, 357
 - isOnTime, 357
 - mandatoryTimes, 357
 - postAdjustValues, 357
 - postAdjustValuesImpl, 358
 - preAdjustValues, 357
 - preAdjustValuesImpl, 357
 - reset, 357
- QuantLib::DiscretizedDiscountBond, 359
- QuantLib::DiscretizedDiscountBond
 - mandatoryTimes, 359
 - reset, 359
- QuantLib::DiscretizedOption, 360
- QuantLib::DiscretizedOption
 - mandatoryTimes, 360
 - postAdjustValuesImpl, 361
 - reset, 360
- QuantLib::Disposable, 362
- QuantLib::DividendVanillaOption, 363
- QuantLib::DividendVanillaOption
 - setupArguments, 364
- QuantLib::DividendVanillaOption::arguments, 365
- QuantLib::DividendVanillaOption::engine, 366
- QuantLib::DKKCurrency, 367
- QuantLib::DKKLibor, 368
- QuantLib::DMinus, 369
- QuantLib::DownRounding, 370
- QuantLib::DPlus, 371
- QuantLib::DPlusDMinus, 372
- QuantLib::DriftTermStructure, 373
- QuantLib::Duration, 375
- QuantLib::DZero, 376
- QuantLib::EarlyExercise, 377
- QuantLib::EEKCurrency, 378
- QuantLib::EndCriteria, 379
- QuantLib::EqualJumpsBinomialTree, 381
- QuantLib::EqualProbabilitiesBinomialTree, 382
- QuantLib::Error, 383
- QuantLib::Error
 - ~Error, 383
 - Error, 383
- QuantLib::ErrorFunction, 384
- QuantLib::ESPCurrency, 385
- QuantLib::EulerDiscretization, 386
- QuantLib::EulerDiscretization
 - covariance, 387
 - diffusion, 386
 - drift, 386
 - variance, 387
- QuantLib::EURCurrency, 388
- QuantLib::Euribor, 389
- QuantLib::EURLibor, 390
- QuantLib::EuropeanExercise, 391
- QuantLib::EuropeanOption, 392
- QuantLib::ExchangeRate, 393
 - Derived, 394
 - Direct, 394
- QuantLib::ExchangeRate
 - ExchangeRate, 394
 - Type, 394
- QuantLib::ExchangeRateManager, 395
- QuantLib::ExchangeRateManager
 - add, 395
 - lookup, 395
- QuantLib::Exercise, 397
- QuantLib::ExplicitEuler, 398
- QuantLib::ExtendedCoxIngersollRoss, 400
- QuantLib::ExtendedCoxIngersollRoss::Dynamics, 402
- QuantLib::ExtendedCoxIngersollRoss::FittingParameter, 403
- QuantLib::ExtendedDiscountCurve, 404
- QuantLib::ExtendedDiscountCurve
 - compoundForwardImpl, 405
 - update, 405
 - zeroYieldImpl, 405
- QuantLib::Extrapolator, 406
- QuantLib::Factorial, 407
- QuantLib::FalsePosition, 408
- QuantLib::FaureRsg, 409
- QuantLib::FDAmericanEngine, 410
- QuantLib::FDBermudanEngine, 411
- QuantLib::FDDividendAmericanEngine, 412
- QuantLib::FDDividendEngine, 413
- QuantLib::FDDividendEuropeanEngine, 414
- QuantLib::FDDividendShoutEngine, 415
- QuantLib::FDEuropeanEngine, 416
- QuantLib::FDShoutEngine, 417
- QuantLib::FDStepConditionEngine, 418
- QuantLib::FDVanillaEngine, 419
- QuantLib::FIMCurrency, 421
- QuantLib::FiniteDifferenceModel, 422
- QuantLib::FiniteDifferenceModel
 - rollback, 422
- QuantLib::FixedCouponBond, 423
- QuantLib::FixedCouponBondHelper, 424
- QuantLib::FixedCouponBondHelper
 - latestDate, 425
 - setTermStructure, 425
- QuantLib::FixedRateCoupon, 426
- QuantLib::FixedRateCoupon

- amount, [427](#)
- QuantLib::FlatForward, [428](#)
- QuantLib::FlatForward
 - update, [429](#)
- QuantLib::FloatingRateBond, [430](#)
- QuantLib::FloatingRateCoupon, [431](#)
- QuantLib::Floor, [433](#)
- QuantLib::FloorTruncation, [434](#)
- QuantLib::ForwardEngine, [435](#)
- QuantLib::ForwardFlat, [436](#)
- QuantLib::ForwardFlatInterpolation, [437](#)
- QuantLib::ForwardFlatInterpolation
 - ForwardFlatInterpolation, [437](#)
- QuantLib::ForwardOptionArguments, [438](#)
- QuantLib::ForwardPerformanceEngine, [439](#)
- QuantLib::ForwardRate, [440](#)
- QuantLib::ForwardRateStructure, [441](#)
- QuantLib::ForwardRateStructure
 - discountImpl, [441](#)
 - zeroYieldImpl, [442](#)
- QuantLib::ForwardSpreadedTermStructure, [443](#)
- QuantLib::ForwardSpreadedTermStructure
 - zeroYieldImpl, [444](#)
- QuantLib::ForwardVanillaOption, [445](#)
- QuantLib::ForwardVanillaOption
 - performCalculations, [446](#)
 - setupArguments, [446](#)
- QuantLib::FraRateHelper, [447](#)
- QuantLib::FraRateHelper
 - latestDate, [448](#)
 - setTermStructure, [447](#)
- QuantLib::FRFCurrency, [449](#)
- QuantLib::FuturesRateHelper, [450](#)
- QuantLib::FuturesRateHelper
 - latestDate, [450](#)
- QuantLib::G2, [451](#)
- QuantLib::G2::FittingParameter, [453](#)
- QuantLib::G2SwaptionEngine, [454](#)
- QuantLib::GammaFunction, [455](#)
- QuantLib::GapPayoff, [456](#)
- QuantLib::GaussChebyshev2thIntegration, [457](#)
- QuantLib::GaussChebyshevIntegration, [458](#)
- QuantLib::GaussGegenbauerIntegration, [459](#)
- QuantLib::GaussHermiteIntegration, [460](#)
- QuantLib::GaussHermitePolynomial, [461](#)
- QuantLib::GaussHyperbolicIntegration, [462](#)
- QuantLib::GaussHyperbolicPolynomial, [463](#)
- QuantLib::GaussianOrthogonalPolynomial, [464](#)
- QuantLib::GaussianQuadrature, [465](#)
- QuantLib::GaussianStatistics, [466](#)
- QuantLib::GaussianStatistics
 - gaussianDownsideDeviation, [466](#)
 - gaussianDownsideVariance, [466](#)
 - gaussianExpectedShortfall, [467](#)
 - gaussianPercentile, [467](#)
 - gaussianPotentialUpside, [467](#)
 - gaussianRegret, [467](#)
 - gaussianTopPercentile, [467](#)
 - gaussianValueAtRisk, [467](#)
- QuantLib::GaussJacobiIntegration, [469](#)
- QuantLib::GaussJacobiPolynomial, [470](#)
- QuantLib::GaussLaguerreIntegration, [471](#)
- QuantLib::GaussLaguerrePolynomial, [472](#)
- QuantLib::GaussLegendreIntegration, [473](#)
- QuantLib::GBPCurrency, [474](#)
- QuantLib::GBPLibor, [475](#)
- QuantLib::GeneralStatistics, [476](#)
- QuantLib::GeneralStatistics
 - add, [478](#)
 - errorEstimate, [477](#)
 - expectationValue, [478](#)
 - kurtosis, [477](#)
 - max, [478](#)
 - mean, [477](#)
 - min, [477](#)
 - percentile, [478](#)
 - skewness, [477](#)
 - standardDeviation, [477](#)
 - topPercentile, [478](#)
 - variance, [477](#)
- QuantLib::GenericEngine, [479](#)
- QuantLib::GenericModelEngine, [480](#)
- QuantLib::GenericModelEngine
 - update, [480](#)
- QuantLib::GenericRiskStatistics, [481](#)
- QuantLib::GenericRiskStatistics
 - averageShortfall, [483](#)
 - downsideDeviation, [482](#)
 - downsideVariance, [481](#)
 - expectedShortfall, [482](#)
 - potentialUpside, [482](#)
 - regret, [482](#)
 - semiDeviation, [481](#)
 - semiVariance, [481](#)
 - shortfall, [482](#)
 - valueAtRisk, [482](#)
- QuantLib::GeometricBrownianMotionProcess, [484](#)
- QuantLib::Germany, [485](#)
 - Eurex, [487](#)
 - FrankfurtStockExchange, [487](#)
 - Settlement, [487](#)
 - Xetra, [487](#)
- QuantLib::Germany

- Market, [487](#)
- QuantLib::GRDCurrency, [488](#)
- QuantLib::Greeks, [489](#)
- QuantLib::HaltonRsg, [490](#)
- QuantLib::Handle, [491](#)
- QuantLib::Handle
 - Handle, [491](#)
 - linkTo, [491](#)
- QuantLib::Helsinki, [492](#)
- QuantLib::HestonModel, [493](#)
- QuantLib::HestonModelHelper, [494](#)
- QuantLib::HestonProcess, [495](#)
- QuantLib::HestonProcess
 - apply, [496](#)
 - time, [496](#)
- QuantLib::History, [497](#)
- QuantLib::History
 - History, [498](#), [499](#)
- QuantLib::History::const_iterator, [500](#)
- QuantLib::History::Entry, [501](#)
- QuantLib::HKDCurrency, [502](#)
- QuantLib::HongKong, [503](#)
- QuantLib::HUFCurrency, [504](#)
- QuantLib::HullWhite, [505](#)
- QuantLib::HullWhite::Dynamics, [507](#)
- QuantLib::HullWhite::FittingParameter, [508](#)
- QuantLib::IEPCurrency, [509](#)
- QuantLib::ILSCurrency, [510](#)
- QuantLib::IMM, [511](#)
- QuantLib::ImplicitEuler, [512](#)
- QuantLib::ImpliedTermStructure, [513](#)
- QuantLib::ImpliedVolTermStructure, [515](#)
- QuantLib::InArrearIndexedCoupon, [517](#)
- QuantLib::IncrementalStatistics, [519](#)
- QuantLib::IncrementalStatistics
 - add, [521](#)
 - addSequence, [521](#)
 - downsideDeviation, [520](#)
 - downsideVariance, [520](#)
 - errorEstimate, [520](#)
 - kurtosis, [521](#)
 - max, [521](#)
 - mean, [520](#)
 - min, [521](#)
 - skewness, [521](#)
 - standardDeviation, [520](#)
 - variance, [520](#)
- QuantLib::Index, [522](#)
- QuantLib::Index
 - fixing, [522](#)
 - name, [522](#)
- QuantLib::IndexedCoupon, [523](#)
- QuantLib::IndexedCoupon
 - amount, [524](#)
 - update, [524](#)
- QuantLib::IndexManager, [525](#)
- QuantLib::INRCurrency, [526](#)
- QuantLib::Instrument, [527](#)
- QuantLib::Instrument
 - calculate, [528](#)
 - performCalculations, [528](#)
 - setPricingEngine, [528](#)
 - setupArguments, [528](#)
 - setupExpired, [528](#)
- QuantLib::IntegralEngine, [530](#)
- QuantLib::InterestRate, [531](#)
- QuantLib::InterestRate
 - compoundFactor, [532](#)
 - discountFactor, [532](#)
 - equivalentRate, [533](#)
 - impliedRate, [533](#)
- QuantLib::InterpolatedDiscountCurve, [534](#)
- QuantLib::InterpolatedForwardCurve, [536](#)
- QuantLib::InterpolatedForwardCurve
 - zeroYieldImpl, [537](#)
- QuantLib::InterpolatedZeroCurve, [538](#)
- QuantLib::Interpolation, [540](#)
- QuantLib::Interpolation2D, [541](#)
- QuantLib::Interpolation2D::templateImpl, [542](#)
- QuantLib::Interpolation2DImpl, [543](#)
- QuantLib::Interpolation::templateImpl, [544](#)
- QuantLib::InterpolationImpl, [545](#)
- QuantLib::InverseCumulativeNormal, [546](#)
- QuantLib::InverseCumulativePoisson, [547](#)
- QuantLib::InverseCumulativeRng, [548](#)
- QuantLib::InverseCumulativeRsg, [549](#)
- QuantLib::IQDCurrency, [550](#)
- QuantLib::IRRCurrency, [551](#)
- QuantLib::ISKCurrency, [552](#)
- QuantLib::Istanbul, [553](#)
- QuantLib::Italy, [554](#)
 - Exchange, [555](#)
 - Settlement, [555](#)
- QuantLib::Italy
 - Market, [555](#)
- QuantLib::ITLCurrency, [556](#)
- QuantLib::JamshidianSwaptionEngine, [557](#)
- QuantLib::JarrowRudd, [558](#)
- QuantLib::Jibar, [559](#)
- QuantLib::Johannesburg, [560](#)
- QuantLib::JointCalendar, [561](#)
- QuantLib::JPYCurrency, [562](#)
- QuantLib::JPYLibor, [563](#)
- QuantLib::JumpDiffusionEngine, [564](#)
- QuantLib::JuQuadraticApproximationEngine, [565](#)
- QuantLib::KnuthUniformRng, [566](#)

- QuantLib::KnuthUniformRng
 - KnuthUniformRng, [566](#)
 - next, [566](#)
- QuantLib::KronrodIntegral, [567](#)
- QuantLib::KRWCurrency, [568](#)
- QuantLib::KWDCurrency, [569](#)
- QuantLib::Lattice, [570](#)
- QuantLib::Lattice
 - partialRollback, [571](#)
 - rollback, [571](#)
- QuantLib::Lattice1D, [572](#)
- QuantLib::Lattice2D, [573](#)
- QuantLib::LatticeShortRateModelEngine, [574](#)
- QuantLib::LatticeShortRateModelEngine
 - update, [574](#)
- QuantLib::LazyObject, [575](#)
- QuantLib::LazyObject
 - calculate, [576](#)
 - freeze, [576](#)
 - performCalculations, [576](#)
 - recalculate, [576](#)
 - unfreeze, [576](#)
 - update, [575](#)
- QuantLib::LeastSquareFunction, [577](#)
- QuantLib::LeastSquareProblem, [578](#)
- QuantLib::LeastSquareProblem
 - targetValueAndGradient, [578](#)
- QuantLib::LecuyerUniformRng, [579](#)
- QuantLib::LecuyerUniformRng
 - LecuyerUniformRng, [579](#)
 - next, [579](#)
- QuantLib::LeisenReimer, [580](#)
- QuantLib::LexicographicalView, [581](#)
- QuantLib::Libor, [583](#)
- QuantLib::Linear, [584](#)
- QuantLib::LinearInterpolation, [585](#)
- QuantLib::LinearInterpolation
 - LinearInterpolation, [585](#)
- QuantLib::LineSearch, [586](#)
- QuantLib::Link, [588](#)
- QuantLib::Link
 - Link, [588](#)
 - linkTo, [589](#)
- QuantLib::LocalConstantVol, [590](#)
- QuantLib::LocalVolCurve, [591](#)
- QuantLib::LocalVolCurve
 - localVolImpl, [592](#)
- QuantLib::LocalVolSurface, [593](#)
- QuantLib::LocalVolTermStructure, [595](#)
- QuantLib::LocalVolTermStructure
 - LocalVolTermStructure, [596](#)
- QuantLib::LogLinear, [597](#)
- QuantLib::LogLinearInterpolation, [598](#)
- QuantLib::LogLinearInterpolation
 - LogLinearInterpolation, [598](#)
- QuantLib::LTLCurrency, [599](#)
- QuantLib::LUTCurrency, [600](#)
- QuantLib::LVLCurrency, [601](#)
- QuantLib::MakeMCDigitalEngine, [602](#)
- QuantLib::MakeMCEuropeanEngine, [603](#)
- QuantLib::MakeMCEuropeanHestonEngine, [604](#)
- QuantLib::MakeSchedule, [605](#)
- QuantLib::Matrix, [606](#)
- QuantLib::Matrix
 - operator+=, [608](#)
 - pseudoSqrt, [608](#)
 - rankReducedSqrt, [609](#)
- QuantLib::MCAmericanBasketEngine, [610](#)
- QuantLib::MCBarrierEngine, [611](#)
- QuantLib::MCBasketEngine, [613](#)
- QuantLib::McCliquetOption, [615](#)
- QuantLib::MCDigitalEngine, [616](#)
- QuantLib::MCDiscreteArithmeticAPEngine, [617](#)
- QuantLib::MCDiscreteArithmeticASO, [619](#)
- QuantLib::MCDiscreteAveragingAsianEngine, [620](#)
- QuantLib::MCDiscreteGeometricAPEngine, [622](#)
- QuantLib::MCEuropeanEngine, [623](#)
- QuantLib::MCEuropeanHestonEngine, [624](#)
- QuantLib::McEverest, [625](#)
- QuantLib::MCHestonEngine, [626](#)
- QuantLib::McHimalaya, [627](#)
- QuantLib::McMaxBasket, [628](#)
- QuantLib::McPagoda, [629](#)
- QuantLib::McPerformanceOption, [630](#)
- QuantLib::McPricer, [631](#)
- QuantLib::McSimulation, [632](#)
- QuantLib::McSimulation
 - calculate, [633](#)
- QuantLib::MCVanillaEngine, [634](#)
- QuantLib::MersenneTwisterUniformRng, [636](#)
- QuantLib::MersenneTwisterUniformRng
 - MersenneTwisterUniformRng, [636](#)
 - next, [636](#)
- QuantLib::Merton76Process, [637](#)
- QuantLib::Merton76Process
 - apply, [638](#)
 - time, [638](#)
- QuantLib::MixedScheme, [639](#)
- QuantLib::Money, [641](#)
 - AutomatedConversion, [642](#)
 - BaseCurrencyConversion, [642](#)
 - NoConversion, [642](#)

- QuantLib::Money
 - ConversionType, 642
- QuantLib::MonotonicCubicSpline, 643
- QuantLib::MonotonicCubicSpline
 - MonotonicCubicSpline, 643
- QuantLib::MonteCarloModel, 644
- QuantLib::MoreGreeks, 645
- QuantLib::MoroInverseCumulativeNormal, 646
- QuantLib::MTLCurrency, 647
- QuantLib::MultiAsset, 648
- QuantLib::MultiAssetOption, 649
- QuantLib::MultiAssetOption
 - performCalculations, 650
 - setupArguments, 650
 - setupExpired, 650
- QuantLib::MultiAssetOption::arguments, 651
- QuantLib::MultiAssetOption::results, 652
- QuantLib::MultiCubicSpline, 653
- QuantLib::MultiPath, 654
- QuantLib::MultiPathGenerator, 655
- QuantLib::MultiVariate, 656
- QuantLib::MXNCurrency, 657
- QuantLib::NaturalCubicSpline, 658
- QuantLib::NaturalCubicSpline
 - NaturalCubicSpline, 658
- QuantLib::NaturalMonotonicCubicSpline, 659
- QuantLib::NaturalMonotonicCubicSpline
 - NaturalMonotonicCubicSpline, 659
- QuantLib::NeumannBC, 660
- QuantLib::NeumannBC
 - applyAfterApplying, 660
 - applyAfterSolving, 661
 - applyBeforeApplying, 660
 - applyBeforeSolving, 660
 - setTime, 661
- QuantLib::Newton, 662
- QuantLib::NewtonSafe, 663
- QuantLib::NLGCurrency, 664
- QuantLib::NoConstraint, 665
- QuantLib::NOKCurrency, 666
- QuantLib::NonLinearLeastSquare, 667
- QuantLib::NormalDistribution, 668
- QuantLib::NPRCurrency, 669
- QuantLib::Null, 670
- QuantLib::NullCalendar, 671
- QuantLib::NullCondition, 672
- QuantLib::NullParameter, 673
- QuantLib::NumericalMethod, 674
- QuantLib::NumericalMethod
 - partialRollback, 674
 - rollback, 674
- QuantLib::NZDCurrency, 676
- QuantLib::NZDLibor, 677
- QuantLib::Observable, 678
- QuantLib::Observable
 - notifyObservers, 678
- QuantLib::ObservableValue, 679
- QuantLib::Observer, 680
- QuantLib::Observer
 - update, 681
- QuantLib::OneAssetOption, 682
- QuantLib::OneAssetOption
 - impliedVolatility, 683
 - performCalculations, 684
 - setupArguments, 683
 - setupExpired, 683
- QuantLib::OneAssetOption::arguments, 685
- QuantLib::OneAssetOption::results, 686
- QuantLib::OneAssetStrikedOption, 687
- QuantLib::OneAssetStrikedOption
 - performCalculations, 688
 - setupArguments, 687
- QuantLib::OneDayCounter, 689
- QuantLib::OneFactorAffineModel, 690
- QuantLib::OneFactorModel, 691
- QuantLib::OneFactorModel::ShortRateDynamics, 692
- QuantLib::OneFactorModel::ShortRateTree, 693
- QuantLib::OneFactorOperator, 694
- QuantLib::OptimizationMethod, 695
- QuantLib::Option, 697
- QuantLib::Option::arguments, 698
- QuantLib::OrnsteinUhlenbeckProcess, 699
- QuantLib::OrnsteinUhlenbeckProcess
 - expectation, 699
 - stdDeviation, 699
 - variance, 700
- QuantLib::Oslo, 701
- QuantLib::Parameter, 702
- QuantLib::ParameterImpl, 703
- QuantLib::ParCoupon, 704
- QuantLib::ParCoupon
 - amount, 705
 - update, 705
- QuantLib::Path, 706
- QuantLib::PathGenerator, 707
- QuantLib::PathPricer, 708
- QuantLib::Payoff, 709
- QuantLib::PercentageStrikePayoff, 710
- QuantLib::Period, 711
- QuantLib::PiecewiseConstantParameter, 712
- QuantLib::PiecewiseYieldCurve, 713
- QuantLib::PiecewiseYieldCurve

- update, [714](#)
- QuantLib::PKRCurrency, [715](#)
- QuantLib::PlainVanillaPayoff, [716](#)
- QuantLib::PLNCurrency, [717](#)
- QuantLib::PoissonDistribution, [718](#)
- QuantLib::PositiveConstraint, [719](#)
- QuantLib::Prague, [720](#)
- QuantLib::PricingEngine, [721](#)
- QuantLib::PrimeNumbers, [722](#)
- QuantLib::Problem, [723](#)
- QuantLib::PTECurrency, [724](#)
- QuantLib::QuantoEngine, [725](#)
- QuantLib::QuantoEngine
 - underlyingArgs, [726](#)
- QuantLib::QuantoForwardVanillaOption, [727](#)
- QuantLib::QuantoForwardVanillaOption
 - setupArguments, [728](#)
- QuantLib::QuantoOptionArguments, [729](#)
- QuantLib::QuantoOptionResults, [730](#)
- QuantLib::QuantoTermStructure, [731](#)
- QuantLib::QuantoVanillaOption, [733](#)
- QuantLib::QuantoVanillaOption
 - performCalculations, [734](#)
 - setupArguments, [734](#)
 - setupExpired, [734](#)
- QuantLib::Quote, [735](#)
- QuantLib::RandomizedLDS, [736](#)
- QuantLib::RandomizedLDS
 - nextRandomizer, [737](#)
- QuantLib::RandomSequenceGenerator, [738](#)
- QuantLib::RateHelper, [739](#)
- QuantLib::RateHelper
 - latestDate, [740](#)
 - setTermStructure, [740](#)
 - update, [740](#)
- QuantLib::Results, [741](#)
- QuantLib::Ridder, [742](#)
- QuantLib::Riyadh, [743](#)
- QuantLib::ROLCurrency, [744](#)
- QuantLib::Rounding, [745](#)
 - Ceiling, [746](#)
 - Closest, [746](#)
 - Down, [746](#)
 - Floor, [746](#)
 - None, [746](#)
 - Up, [746](#)
- QuantLib::Rounding
 - Rounding, [746](#)
 - Type, [745](#)
- QuantLib::SalvagingAlgorithm, [747](#)
- QuantLib::Sample, [748](#)
- QuantLib::SampledCurve, [749](#)
- QuantLib::SARCurrency, [750](#)
- QuantLib::Schedule, [751](#)
- QuantLib::Secant, [752](#)
- QuantLib::SeedGenerator, [753](#)
- QuantLib::SegmentIntegral, [754](#)
- QuantLib::SEKCurrency, [755](#)
- QuantLib::Seoul, [756](#)
- QuantLib::SequenceStatistics, [757](#)
- QuantLib::Settings, [759](#)
- QuantLib::Settings
 - evaluationDate, [759](#)
- QuantLib::SGDCurrency, [761](#)
- QuantLib::Short, [762](#)
- QuantLib::Short
 - amount, [762](#)
- QuantLib::Short< ParCoupon >, [763](#)
- QuantLib::ShortRateModel, [764](#)
- QuantLib::ShortRateModel
 - calibrate, [765](#)
 - update, [765](#)
- QuantLib::ShoutCondition, [766](#)
- QuantLib::SimpleCashFlow, [767](#)
- QuantLib::SimpleCashFlow
 - amount, [767](#)
- QuantLib::SimpleDayCounter, [768](#)
- QuantLib::SimpleQuote, [769](#)
- QuantLib::SimpleSwap, [770](#)
- QuantLib::SimpleSwap
 - setupArguments, [771](#)
- QuantLib::SimpleSwap::arguments, [772](#)
- QuantLib::SimpleSwap::results, [773](#)
- QuantLib::Simplex, [774](#)
- QuantLib::Simplex
 - Simplex, [774](#)
- QuantLib::SimpsonIntegral, [775](#)
- QuantLib::Singapore, [776](#)
- QuantLib::SingleAsset, [777](#)
- QuantLib::SingleAssetOption, [778](#)
- QuantLib::SingleAssetOption
 - impliedVolatility, [779](#)
- QuantLib::Singleton, [780](#)
- QuantLib::SingleVariate, [781](#)
- QuantLib::SITCurrency, [782](#)
- QuantLib::SKKCurrency, [783](#)
- QuantLib::SobolRsg, [784](#)
- QuantLib::SobolRsg
 - SobolRsg, [785](#)
- QuantLib::Solver1D, [786](#)
- QuantLib::Solver1D
 - setMaxEvaluations, [787](#)
 - solve, [787](#)
- QuantLib::SquareRootProcess, [788](#)
- QuantLib::StatsHolder, [789](#)
- QuantLib::SteepestDescent, [790](#)
- QuantLib::step_iterator, [791](#)

- QuantLib::StepCondition, 792
- QuantLib::StepConditionSet, 793
- QuantLib::StochasticProcess, 794
- QuantLib::StochasticProcess
 - apply, 796
 - covariance, 795
 - evolve, 795
 - expectation, 795
 - stdDeviation, 795
 - time, 796
 - update, 796
- QuantLib::StochasticProcess1D, 797
- QuantLib::StochasticProcess1D
 - apply, 798
 - evolve, 798
 - expectation, 798
 - stdDeviation, 798
 - variance, 798
- QuantLib::StochasticProcess1D::discretization, 799
- QuantLib::StochasticProcess::discretization, 800
- QuantLib::StochasticProcessArray, 801
- QuantLib::StochasticProcessArray
 - apply, 802
 - covariance, 802
 - expectation, 802
 - stdDeviation, 802
 - time, 802
- QuantLib::Stock, 803
- QuantLib::Stock
 - performCalculations, 803
- QuantLib::Stockholm, 804
- QuantLib::StrikedTypePayoff, 805
- QuantLib::StulzEngine, 806
- QuantLib::SuperSharePayoff, 807
- QuantLib::SVD, 808
- QuantLib::Swap, 809
- QuantLib::Swap
 - performCalculations, 810
 - sensitivity, 810
 - setupExpired, 810
- QuantLib::SwapRateHelper, 811
- QuantLib::SwapRateHelper
 - latestDate, 812
 - setTermStructure, 812
- QuantLib::Swaption, 813
- QuantLib::Swaption
 - setupArguments, 814
- QuantLib::Swaption::arguments, 815
- QuantLib::Swaption::results, 816
- QuantLib::SwaptionVolatilityMatrix, 817
- QuantLib::SwaptionVolatilityStructure, 819
- QuantLib::SwaptionVolatilityStructure
 - SwaptionVolatilityStructure, 820
- QuantLib::Sydney, 821
- QuantLib::SymmetricSchurDecomposition, 822
- QuantLib::SymmetricSchurDecomposition
 - SymmetricSchurDecomposition, 822
- QuantLib::TabulatedGaussLegendre, 823
- QuantLib::Taipei, 824
- QuantLib::Taiwan, 825
- QuantLib::TARGET, 826
- QuantLib::TermStructure, 827
- QuantLib::TermStructure
 - TermStructure, 828
 - update, 828
- QuantLib::TermStructureConsistentModel, 829
- QuantLib::TermStructureFittingParameter, 830
- QuantLib::THBCurrency, 831
- QuantLib::Thirty360, 832
- QuantLib::Tian, 833
- QuantLib::Tibor, 834
- QuantLib::TimeBasket, 835
- QuantLib::TimeGrid, 836
- QuantLib::TimeGrid
 - TimeGrid, 837
- QuantLib::Tokyo, 838
- QuantLib::Toronto, 840
- QuantLib::TqrEigenDecomposition, 841
- QuantLib::TrapezoidIntegral, 842
- QuantLib::Tree, 844
- QuantLib::TreeCapFloorEngine, 845
- QuantLib::TreeSwaptionEngine, 846
- QuantLib::TridiagonalOperator, 847
- QuantLib::TridiagonalOperator::TimeSetter, 849
- QuantLib::Trigeorgis, 850
- QuantLib::TrinomialTree, 851
- QuantLib::TRLCurrency, 852
- QuantLib::TRLibor, 853
- QuantLib::TRYCurrency, 854
- QuantLib::TTDCurrency, 855
- QuantLib::TWDCurrency, 856
- QuantLib::TwoFactorModel, 857
- QuantLib::TwoFactorModel::ShortRateDynamics, 858
- QuantLib::TwoFactorModel::ShortRateTree, 859
- QuantLib::TypePayoff, 860
- QuantLib::UnitedKingdom, 861
 - Exchange, 862
 - Settlement, 862
- QuantLib::UnitedKingdom
 - Market, 862

- QuantLib::UnitedStates, 863
 - Exchange, 865
 - Settlement, 865
- QuantLib::UnitedStates
 - Market, 865
- QuantLib::UpFrontIndexedCoupon, 866
- QuantLib::UpRounding, 867
- QuantLib::USDCurrency, 868
- QuantLib::USDLibor, 869
- QuantLib::Value, 870
- QuantLib::VanillaOption, 871
- QuantLib::VanillaOption::engine, 872
- QuantLib::Vasicek, 873
- QuantLib::Vasicek::Dynamics, 875
- QuantLib::VEBCurrency, 876
- QuantLib::Visitor, 877
- QuantLib::Warsaw, 878
- QuantLib::Wellington, 879
- QuantLib::Xibor, 880
- QuantLib::Xibor
 - fixing, 881
 - name, 881
 - update, 881
- QuantLib::YieldTermStructure, 882
- QuantLib::YieldTermStructure
 - discount, 884
 - forwardRate, 884
 - parRate, 884, 885
 - YieldTermStructure, 884
 - zeroRate, 884
- QuantLib::ZARCurrency, 886
- QuantLib::ZeroCouponBond, 887
- QuantLib::ZeroSpreadedTermStructure, 888
- QuantLib::ZeroSpreadedTermStructure
 - forwardImpl, 889
- QuantLib::ZeroYield, 890
- QuantLib::ZeroYieldStructure, 891
- QuantLib::ZeroYieldStructure
 - discountImpl, 891
- QuantLib::Zibor, 893
- QuantLib::Zurich, 894
- Quanto option engines, 102
- Quarterly
 - datetime, 91
- rankReducedSqrt
 - QuantLib::Matrix, 609
- recalculate
 - QuantLib::LazyObject, 576
- regret
 - QuantLib::GenericRiskStatistics, 482
- removeHoliday
 - QuantLib::Calendar, 255
- reset
 - QuantLib::DiscretizedAsset, 357
 - QuantLib::DiscretizedDiscountBond, 359
 - QuantLib::DiscretizedOption, 360
- rollback
 - QuantLib::FiniteDifferenceModel, 422
 - QuantLib::Lattice, 571
 - QuantLib::NumericalMethod, 674
- Rounding
 - QuantLib::Rounding, 746
- SecondDerivative
 - QuantLib::CubicSpline, 325
- Semiannual
 - datetime, 91
- semiDeviation
 - QuantLib::GenericRiskStatistics, 481
- semiVariance
 - QuantLib::GenericRiskStatistics, 481
- sensitivity
 - QuantLib::Swap, 810
- sequencestatistics.hpp
 - DEFINE_SEQUENCE_STAT_CONST_-METHOD_DOUBLE, 1093
 - DEFINE_SEQUENCE_STAT_CONST_-METHOD_VOID, 1093
- setMaxEvaluations
 - QuantLib::Solver1D, 787
- setPricingEngine
 - QuantLib::Instrument, 528
- setTermStructure
 - QuantLib::DepositRateHelper, 344
 - QuantLib::FixedCouponBondHelper, 425
 - QuantLib::FraRateHelper, 447
 - QuantLib::RateHelper, 740
 - QuantLib::SwapRateHelper, 812
- setTime
 - QuantLib::BoundaryCondition, 236
 - QuantLib::DirichletBC, 348
 - QuantLib::NeumannBC, 661
- Settlement
 - QuantLib::Germany, 487
 - QuantLib::Italy, 555
 - QuantLib::UnitedKingdom, 862
 - QuantLib::UnitedStates, 865
- setupArguments
 - QuantLib::BarrierOption, 181
 - QuantLib::BasketOption, 185
 - QuantLib::CapFloor, 263
 - QuantLib::CliquetOption, 286
 - QuantLib::ContinuousAveraging-AsianOption, 303

- QuantLib::ConvertibleBond::option, 307
- QuantLib::DiscreteAveragingAsian-Option, 352
- QuantLib::DividendVanillaOption, 364
- QuantLib::ForwardVanillaOption, 446
- QuantLib::Instrument, 528
- QuantLib::MultiAssetOption, 650
- QuantLib::OneAssetOption, 683
- QuantLib::OneAssetStrikedOption, 687
- QuantLib::QuantoForwardVanilla-Option, 728
- QuantLib::QuantoVanillaOption, 734
- QuantLib::SimpleSwap, 771
- QuantLib::Swaption, 814
- setupExpired
 - QuantLib::Instrument, 528
 - QuantLib::MultiAssetOption, 650
 - QuantLib::OneAssetOption, 683
 - QuantLib::QuantoVanillaOption, 734
 - QuantLib::Swap, 810
- Short-rate modelling framework, 112
- shortfall
 - QuantLib::GenericRiskStatistics, 482
- Side
 - QuantLib::BoundaryCondition, 235
- Simplex
 - QuantLib::Simplex, 774
- skewness
 - QuantLib::GeneralStatistics, 477
 - QuantLib::IncrementalStatistics, 521
- SobolRsg
 - QuantLib::SobolRsg, 785
- solve
 - QuantLib::Solver1D, 787
- standardDeviation
 - QuantLib::GeneralStatistics, 477
 - QuantLib::IncrementalStatistics, 520
- standardDeviations
 - QuantLib::CovarianceDecomposition, 316
- stdDeviation
 - QuantLib::OrnsteinUhlenbeckProcess, 699
 - QuantLib::StochasticProcess, 795
 - QuantLib::StochasticProcess1D, 798
 - QuantLib::StochasticProcessArray, 802
- Swaption engines, 103
- SwaptionVolatilityStructure
 - QuantLib::SwaptionVolatilityStructure, 820
- SymmetricSchurDecomposition
 - QuantLib::SymmetricSchur-Decomposition, 822
- targetValueAndGradient
 - QuantLib::LeastSquareProblem, 578
- Template capabilities, 137
- templateMacros
 - QL_TYPENAME, 137
- Term structures, 130
- TermStructure
 - QuantLib::TermStructure, 828
- time
 - QuantLib::BlackScholesProcess, 218
 - QuantLib::HestonProcess, 496
 - QuantLib::Merton76Process, 638
 - QuantLib::StochasticProcess, 796
 - QuantLib::StochasticProcessArray, 802
- TimeGrid
 - QuantLib::TimeGrid, 837
- topPercentile
 - QuantLib::GeneralStatistics, 478
- Type
 - QuantLib::ExchangeRate, 394
 - QuantLib::Rounding, 745
- Unadjusted
 - datetime, 90
- underlyingArgs
 - QuantLib::QuantoEngine, 726
- unfreeze
 - QuantLib::LazyObject, 576
- Up
 - QuantLib::Rounding, 746
- update
 - QuantLib::AffineTermStructure, 150
 - QuantLib::BlackModel, 214
 - QuantLib::BlackScholesProcess, 218
 - QuantLib::CalibrationHelper, 260
 - QuantLib::CapVolatilityVector, 275
 - QuantLib::CompositeQuote, 295
 - QuantLib::DerivedQuote, 346
 - QuantLib::ExtendedDiscountCurve, 405
 - QuantLib::FlatForward, 429
 - QuantLib::GenericModelEngine, 480
 - QuantLib::IndexedCoupon, 524
 - QuantLib::LatticeShortRateModel-Engine, 574
 - QuantLib::LazyObject, 575
 - QuantLib::Observer, 681
 - QuantLib::ParCoupon, 705
 - QuantLib::PiecewiseYieldCurve, 714
 - QuantLib::RateHelper, 740
 - QuantLib::ShortRateModel, 765
 - QuantLib::StochasticProcess, 796
 - QuantLib::TermStructure, 828
 - QuantLib::Xibor, 881

Utilities, [132](#)

valueAtRisk

QuantLib::GenericRiskStatistics, [482](#)

Vanilla option engines, [104](#)

variance

QuantLib::EulerDiscretization, [387](#)

QuantLib::GeneralStatistics, [477](#)

QuantLib::IncrementalStatistics, [520](#)

QuantLib::OrnsteinUhlenbeckProcess,
[700](#)

QuantLib::StochasticProcess1D, [798](#)

variances

QuantLib::CovarianceDecomposition,
[316](#)

Weekday

datetime, [91](#)

Xetra

QuantLib::Germany, [487](#)

yield

QuantLib::Bond, [234](#)

YieldTermStructure

QuantLib::YieldTermStructure, [884](#)

yieldtermstructures

DiscountCurve, [131](#)

zeroRate

QuantLib::YieldTermStructure, [884](#)

zeroYieldImpl

QuantLib::CompoundForward, [297](#)

QuantLib::ExtendedDiscountCurve,
[405](#)

QuantLib::ForwardRateStructure, [442](#)

QuantLib::ForwardSpreadTerm-
Structure, [444](#)

QuantLib::InterpolatedForwardCurve,
[537](#)