

QuantLib

An open source library for quantitative finance

Version 0.3.6

Generated by Doxygen 1.3.6

15 Apr 2004

Contents

I	User Manual	1
1	An introduction to QuantLib	3
1.1	Introduction	3
1.2	Project overview	4
1.3	Where to get QuantLib	14
1.4	Installation	15
1.5	Usage	16
1.6	Version history	17
1.7	Additional resources	30
1.8	The QuantLib Group	31
1.9	QuantLib License	32
2	QuantLib components	33
2.1	Core classes	33
2.2	Date and time calculations	34
2.3	Lattice methods	35
2.4	The finite differences framework	37
2.5	The Monte Carlo framework	42
2.6	The short-rate modelling framework	48
2.7	Currencies and FX rates	50
2.8	Instruments and pricers	51
2.9	Math tools	53
2.10	Design patterns	55
2.11	Term structures	56
2.12	Utilities	57
3	Examples	59

II	Reference Manual	61
4	QuantLib Module Index	63
4.1	QuantLib Modules	63
5	QuantLib Hierarchical Index	65
5.1	QuantLib Class Hierarchy	65
6	QuantLib Class Index	77
6.1	QuantLib Class List	77
7	QuantLib File Index	87
7.1	QuantLib File List	87
8	QuantLib Module Documentation	95
8.1	Global QuantLib macros	95
8.2	Math functions	97
8.3	Numeric limits	98
8.4	Time functions	99
8.5	String functions	100
8.6	Character functions	101
8.7	Input/output functions	102
8.8	Min and max functions	103
8.9	Template capabilities	104
8.10	Iterator support	106
9	QuantLib Class Documentation	107
9.1	Actual360 Class Reference	107
9.2	Actual365 Class Reference	108
9.3	ActualActual Class Reference	109
9.4	AcyclicVisitor Class Reference	110
9.5	AdditiveEQPBinoialTree Class Reference	111
9.6	AffineModel Class Reference	112
9.7	AffineTermStructure Class Reference	113
9.8	AmericanCondition Class Reference	115
9.9	AmericanExercise Class Reference	116
9.10	AmericanPayoffAtExpiry Class Reference	117
9.11	AmericanPayoffAtHit Class Reference	118
9.12	AnalyticalCapFloor Class Reference	119

9.13	AnalyticBarrierEngine Class Reference	120
9.14	AnalyticDigitalAmericanEngine Class Reference	121
9.15	AnalyticDiscreteAveragingAsianEngine Class Reference	122
9.16	AnalyticEuropeanEngine Class Reference	123
9.17	Arguments Class Reference	124
9.18	ArmijoLineSearch Class Reference	125
9.19	Array Class Reference	126
9.20	ArrayFormatter Class Reference	128
9.21	AssertionFailedError Class Reference	129
9.22	AssetOrNothingPayoff Class Reference	130
9.23	AUDLibor Class Reference	131
9.24	Average Struct Reference	132
9.25	BaroneAdesiWhaleyApproximationEngine Class Reference	133
9.26	Barrier Struct Reference	134
9.27	BarrierEngine Class Reference	135
9.28	BarrierOption Class Reference	136
9.29	BarrierOption::arguments Class Reference	138
9.30	BasketEngine Class Reference	139
9.31	BasketOption Class Reference	140
9.32	BasketOption::arguments Class Reference	142
9.33	BermudanExercise Class Reference	143
9.34	BicubicSpline Class Reference	144
9.35	BicubicSpline::Impl Class Template Reference	145
9.36	BilinearInterpolation Class Reference	146
9.37	BilinearInterpolation::Impl Class Template Reference	147
9.38	BinomialDistribution Class Reference	148
9.39	BinomialTree Class Reference	149
9.40	BinomialVanillaEngine Class Template Reference	150
9.41	Bisection Class Reference	151
9.42	BivariateCumulativeNormalDistribution Class Reference	152
9.43	BjerkstrandStenslandApproximationEngine Class Reference	153
9.44	BlackCapFloor Class Reference	154
9.45	BlackConstantVol Class Reference	155
9.46	BlackKarasinski Class Reference	157
9.47	BlackKarasinski::Dynamics Class Reference	158
9.48	BlackModel Class Reference	159

9.49	BlackScholesLattice Class Reference	161
9.50	BlackScholesProcess Class Reference	162
9.51	BlackSwaption Class Reference	163
9.52	BlackVarianceCurve Class Reference	164
9.53	BlackVarianceSurface Class Reference	166
9.54	BlackVarianceTermStructure Class Reference	168
9.55	BlackVolatilityTermStructure Class Reference	169
9.56	BlackVolTermStructure Class Reference	170
9.57	BoundaryCondition Class Template Reference	172
9.58	BoundaryConstraint Class Reference	174
9.59	BoxMullerGaussianRng Class Template Reference	175
9.60	BPSBasketCalculator Class Reference	176
9.61	BPSCalculator Class Reference	177
9.62	Brent Class Reference	178
9.63	Bridge Class Template Reference	179
9.64	BrownianBridge Class Template Reference	180
9.65	BSMOperator Class Reference	181
9.66	Budapest Class Reference	182
9.67	CADLibor Class Reference	183
9.68	Calendar Class Reference	184
9.69	Calendar::WesternImpl Class Reference	187
9.70	CalendarImpl Class Reference	188
9.71	CalibrationHelper Class Reference	189
9.72	CalibrationSet Class Reference	191
9.73	Cap Class Reference	192
9.74	CapFlatVolatilityStructure Class Reference	193
9.75	CapFlatVolatilityVector Class Reference	194
9.76	CapFloor Class Reference	195
9.77	CapFloor::arguments Class Reference	197
9.78	CapFloor::results Class Reference	198
9.79	CapletForwardVolatilityStructure Class Reference	199
9.80	CashFlow Class Reference	200
9.81	CashOrNothingPayoff Class Reference	201
9.82	CHFLibor Class Reference	202
9.83	CLGaussianRng Class Template Reference	203
9.84	CliquetEngine Class Reference	204

9.85	CliquetOption::arguments Class Reference	205
9.86	CliquetOptionPricer Class Reference	206
9.87	Collar Class Reference	207
9.88	combining_iterator Class Template Reference	208
9.89	Composite Class Template Reference	210
9.90	CompositeConstraint Class Reference	211
9.91	CompositeQuote Class Template Reference	212
9.92	ConjugateGradient Class Reference	213
9.93	ConstantParameter Class Reference	214
9.94	Constraint Class Reference	215
9.95	ConstraintImpl Class Reference	216
9.96	ContinuousGeometricAPO Class Reference	217
9.97	Copenhagen Class Reference	218
9.98	CostFunction Class Reference	219
9.99	coupling_iterator Class Template Reference	220
9.100	Coupon Class Reference	222
9.101	CoxIngersollRoss Class Reference	224
9.102	CoxIngersollRoss::Dynamics Class Reference	226
9.103	CoxRossRubinstein Class Reference	227
9.104	CrankNicolson Class Template Reference	228
9.105	Cubic Class Reference	229
9.106	CubicSpline Class Reference	230
9.107	CumulativeBinomialDistribution Class Reference	232
9.108	CumulativeNormalDistribution Class Reference	233
9.109	CumulativePoissonDistribution Class Reference	234
9.110	CuriouslyRecurringTemplate Class Template Reference	235
9.111	CurrencyFormatter Class Reference	236
9.112	Date Class Reference	237
9.113	DateFormatter Class Reference	239
9.114	DayCounter Class Reference	240
9.115	DayCounterImpl Class Reference	242
9.116	DepositRateHelper Class Reference	243
9.117	DerivedQuote Class Template Reference	245
9.118	DiffusionProcess Class Reference	246
9.119	DirichletBC Class Reference	248
9.120	DiscountCurve Class Reference	249

9.121	DiscountStructure Class Reference	251
9.122	DiscrepancyStatistics Class Reference	253
9.123	DiscreteAveragingAsianEngine Class Reference	254
9.124	DiscreteAveragingAsianOption Class Reference	255
9.125	DiscreteAveragingAsianOption::arguments Class Reference	257
9.126	DiscreteGeometricAPO Class Reference	258
9.127	DiscreteGeometricASO Class Reference	259
9.128	DiscretizedAsset Class Reference	260
9.129	DiscretizedDiscountBond Class Reference	262
9.130	DiscretizedOption Class Reference	263
9.131	Disposable Class Template Reference	264
9.132	DMinus Class Reference	265
9.133	DoubleFormatter Class Reference	266
9.134	DPlus Class Reference	267
9.135	DPlusDMinus Class Reference	268
9.136	DriftTermStructure Class Reference	269
9.137	DZero Class Reference	271
9.138	EarlyExercise Class Reference	272
9.139	EndCriteria Class Reference	273
9.140	EqualJumpsBinomialTree Class Reference	275
9.141	EqualProbabilitiesBinomialTree Class Reference	276
9.142	Error Class Reference	277
9.143	ErrorFunction Class Reference	278
9.144	Euribor Class Reference	279
9.145	EuroFormatter Class Reference	280
9.146	EuropeanExercise Class Reference	281
9.147	EuropeanOption Class Reference	282
9.148	Exercise Class Reference	283
9.149	ExplicitEuler Class Template Reference	284
9.150	ExtendedCoxIngersollRoss Class Reference	285
9.151	ExtendedCoxIngersollRoss::Dynamics Class Reference	286
9.152	ExtendedCoxIngersollRoss::FittingParameter Class Reference	287
9.153	ExtendedDiscountCurve Class Reference	288
9.154	Factorial Class Reference	290
9.155	FalsePosition Class Reference	291
9.156	FdAmericanOption Class Reference	292

9.157	FdBermudanOption Class Reference	293
9.158	FdBsmOption Class Reference	294
9.159	FdDividendAmericanOption Class Reference	296
9.160	FdDividendEuropeanOption Class Reference	297
9.161	FdDividendShoutOption Class Reference	298
9.162	FdEuropean Class Reference	299
9.163	FdStepConditionOption Class Reference	300
9.164	filtering_iterator Class Template Reference	301
9.165	FiniteDifferenceModel Class Template Reference	302
9.166	FixedRateCoupon Class Reference	303
9.167	FloatingRateCoupon Class Reference	305
9.168	Floor Class Reference	307
9.169	ForwardEngine Class Template Reference	308
9.170	ForwardOptionArguments Class Template Reference	309
9.171	ForwardPerformanceEngine Class Template Reference	310
9.172	ForwardRateStructure Class Reference	311
9.173	ForwardSpreadedTermStructure Class Reference	313
9.174	ForwardVanillaOption Class Reference	315
9.175	Frankfurt Class Reference	317
9.176	FraRateHelper Class Reference	318
9.177	FuturesRateHelper Class Reference	320
9.178	G2 Class Reference	321
9.179	G2::FittingParameter Class Reference	323
9.180	GammaFunction Class Reference	324
9.181	GapPayoff Class Reference	325
9.182	GaussianStatistics Class Template Reference	326
9.183	GBPLibor Class Reference	328
9.184	GeneralStatistics Class Reference	329
9.185	GenericEngine Class Template Reference	332
9.186	GenericModelEngine Class Template Reference	333
9.187	GenericRiskStatistics Class Template Reference	334
9.188	Greeks Class Reference	337
9.189	HaltonRsg Class Reference	338
9.190	Handle Class Template Reference	339
9.191	Helsinki Class Reference	341
9.192	History Class Reference	342

9.193	History::const_iterator Class Reference	345
9.194	History::Entry Class Reference	346
9.195	HullWhite Class Reference	347
9.196	HullWhite::Dynamics Class Reference	348
9.197	HullWhite::FittingParameter Class Reference	349
9.198	ICGaussianRng Class Template Reference	350
9.199	ICGaussianRsg Class Template Reference	351
9.200	IllegalArgumentError Class Reference	352
9.201	IllegalResultError Class Reference	353
9.202	ImplicitEuler Class Template Reference	354
9.203	ImpliedTermStructure Class Reference	355
9.204	ImpliedVolTermStructure Class Reference	357
9.205	InArrearIndexedCoupon Class Reference	359
9.206	IncrementalStatistics Class Reference	360
9.207	Index Class Reference	363
9.208	IndexedCoupon Class Reference	364
9.209	IndexError Class Reference	366
9.210	Instrument Class Reference	367
9.211	IntegerFormatter Class Reference	370
9.212	IntegralEngine Class Reference	371
9.213	Interpolation Class Reference	372
9.214	Interpolation2D Class Reference	373
9.215	Interpolation2D::templateImpl Class Template Reference	374
9.216	Interpolation2DImpl Class Reference	375
9.217	Interpolation::templateImpl Class Template Reference	376
9.218	InterpolationImpl Class Reference	377
9.219	InverseCumulativeNormal Class Reference	378
9.220	JamshidianSwaption Class Reference	379
9.221	JarrowRudd Class Reference	380
9.222	Johannesburg Class Reference	381
9.223	JointCalendar Class Reference	382
9.224	JPYLibor Class Reference	383
9.225	KnuthUniformRng Class Reference	384
9.226	KronrodIntegral Class Reference	385
9.227	Lattice Class Reference	386
9.228	Lattice2D Class Reference	388

9.229	LatticeShortRateModelEngine Class Template Reference	389
9.230	LazyObject Class Reference	391
9.231	LeastSquareFunction Class Reference	394
9.232	LeastSquareProblem Class Reference	395
9.233	LecuyerUniformRng Class Reference	396
9.234	LeisenReimer Class Reference	397
9.235	LexicographicalView Class Template Reference	398
9.236	Linear Class Reference	400
9.237	LinearInterpolation Class Reference	401
9.238	LineSearch Class Reference	402
9.239	Link Class Template Reference	404
9.240	LocalConstantVol Class Reference	406
9.241	LocalVolCurve Class Reference	408
9.242	LocalVolSurface Class Reference	410
9.243	LocalVolTermStructure Class Reference	412
9.244	LogLinear Class Reference	413
9.245	LogLinearInterpolation Class Reference	414
9.246	London Class Reference	415
9.247	lowest_category_iterator Struct Template Reference	416
9.248	MakeSchedule Class Reference	417
9.249	Matrix Class Reference	418
9.250	MCAmericanBasketEngine Class Reference	422
9.251	MCBarrierEngine Class Template Reference	423
9.252	McBasket Class Reference	425
9.253	McBasketEngine Class Template Reference	426
9.254	McCliquetOption Class Reference	428
9.255	MCDigitalEngine Class Template Reference	429
9.256	McDiscreteArithmeticAPO Class Reference	430
9.257	McDiscreteArithmeticASO Class Reference	431
9.258	MCEuropeanEngine Class Template Reference	432
9.259	McEverest Class Reference	433
9.260	McHimalaya Class Reference	434
9.261	McMaxBasket Class Reference	435
9.262	McPagoda Class Reference	436
9.263	McPerformanceOption Class Reference	437
9.264	McPricer Class Template Reference	438

9.265	McSimulation Class Template Reference	439
9.266	MCVanillaEngine Class Template Reference	441
9.267	MersenneTwisterUniformRng Class Reference	443
9.268	Milan Class Reference	444
9.269	MixedScheme Class Template Reference	445
9.270	MonotonicCubicSpline Class Reference	447
9.271	MonteCarloModel Class Template Reference	448
9.272	MoreGreeks Class Reference	449
9.273	MoroInverseCumulativeNormal Class Reference	450
9.274	MultiAssetOption Class Reference	451
9.275	MultiAssetOption::arguments Class Reference	453
9.276	MultiAssetOption::results Class Reference	454
9.277	MultiPath Class Reference	455
9.278	MultiPathGenerator Class Template Reference	456
9.279	MultiPathGenerator_old Class Template Reference	457
9.280	NaturalCubicSpline Class Reference	458
9.281	NaturalMonotonicCubicSpline Class Reference	459
9.282	NeumannBC Class Reference	460
9.283	Newton Class Reference	461
9.284	NewtonSafe Class Reference	462
9.285	NewYork Class Reference	463
9.286	NoConstraint Class Reference	464
9.287	NonLinearLeastSquare Class Reference	465
9.288	NormalDistribution Class Reference	466
9.289	Null Class Template Reference	467
9.290	NullCalendar Class Reference	468
9.291	NullParameter Class Reference	469
9.292	NumericalMethod Class Reference	470
9.293	Observable Class Reference	471
9.294	Observer Class Reference	473
9.295	OneAssetOption Class Reference	476
9.296	OneAssetOption::arguments Class Reference	479
9.297	OneAssetOption::results Class Reference	480
9.298	OneAssetStrikedOption Class Reference	481
9.299	OneFactorAffineModel Class Reference	483
9.300	OneFactorModel Class Reference	484

9.301	OneFactorModel::ShortRateDynamics Class Reference	485
9.302	OneFactorModel::ShortRateTree Class Reference	486
9.303	OneFactorOperator Class Reference	487
9.304	OptimizationMethod Class Reference	488
9.305	Option Class Reference	490
9.306	Option::arguments Class Reference	491
9.307	OptionTypeFormatter Class Reference	492
9.308	OrnsteinUhlenbeckProcess Class Reference	493
9.309	Oslo Class Reference	495
9.310	OutOfMemoryError Class Reference	496
9.311	Parameter Class Reference	497
9.312	ParameterImpl Class Reference	498
9.313	ParCoupon Class Reference	499
9.314	Path Class Reference	501
9.315	PathGenerator Class Template Reference	502
9.316	PathGenerator_old Class Template Reference	503
9.317	PathPricer Class Template Reference	504
9.318	PathPricer_old Class Template Reference	505
9.319	Payoff Class Reference	506
9.320	PercentageStrikePayoff Class Reference	507
9.321	PerformanceOption Class Reference	508
9.322	Period Class Reference	509
9.323	PiecewiseConstantParameter Class Reference	510
9.324	PiecewiseFlatForward Class Reference	511
9.325	PlainVanillaPayoff Class Reference	513
9.326	PoissonDistribution Class Reference	514
9.327	PositiveConstraint Class Reference	515
9.328	PostconditionNotSatisfiedError Class Reference	516
9.329	PreconditionNotSatisfiedError Class Reference	517
9.330	PricingEngine Class Reference	518
9.331	PrimeNumbers Class Reference	519
9.332	Problem Class Reference	520
9.333	processing_iterator Class Template Reference	521
9.334	QuantoEngine Class Template Reference	523
9.335	QuantoForwardVanillaOption Class Reference	524
9.336	QuantoOptionArguments Class Template Reference	526

9.337	QuantoOptionResults Class Template Reference	527
9.338	QuantoTermStructure Class Reference	528
9.339	QuantoVanillaOption Class Reference	530
9.340	Quote Class Reference	532
9.341	RandomArrayGenerator Class Template Reference	533
9.342	RandomSequenceGenerator Class Template Reference	534
9.343	RateFormatter Class Reference	535
9.344	RateHelper Class Reference	536
9.345	RelinkableHandle Class Template Reference	538
9.346	Results Class Reference	539
9.347	Ridder Class Reference	540
9.348	SalvagingAlgorithm Struct Reference	541
9.349	Sample Struct Template Reference	542
9.350	Schedule Class Reference	543
9.351	Secant Class Reference	544
9.352	SegmentIntegral Class Reference	545
9.353	SequenceStatistics Class Template Reference	546
9.354	Short Class Template Reference	548
9.355	ShortFloatingRateCoupon Class Reference	549
9.356	ShortRateModel Class Reference	550
9.357	ShoutCondition Class Reference	552
9.358	SimpleCashFlow Class Reference	553
9.359	SimpleDayCounter Class Reference	554
9.360	SimpleQuote Class Reference	555
9.361	SimpleSwap Class Reference	556
9.362	SimpleSwap::arguments Class Reference	558
9.363	SimpleSwap::results Class Reference	559
9.364	Simplex Class Reference	560
9.365	SimpsonIntegral Class Reference	561
9.366	SingleAssetOption Class Reference	562
9.367	SobolRsg Class Reference	564
9.368	Solver1D Class Template Reference	565
9.369	SquareRootProcess Class Reference	567
9.370	StatsHolder Class Reference	568
9.371	SteepestDescent Class Reference	569
9.372	StepCondition Class Template Reference	570

9.373	stepping_iterator Class Template Reference	571
9.374	StochasticProcess Class Reference	573
9.375	Stock Class Reference	574
9.376	Stockholm Class Reference	575
9.377	StrikedTypePayoff Class Reference	576
9.378	StringFormatter Class Reference	577
9.379	StulzEngine Class Reference	578
9.380	SuperSharePayoff Class Reference	579
9.381	SVD Class Reference	580
9.382	Swap Class Reference	581
9.383	SwapRateHelper Class Reference	583
9.384	Swaption Class Reference	585
9.385	Swaption::arguments Class Reference	587
9.386	Swaption::results Class Reference	588
9.387	SwaptionVolatilityMatrix Class Reference	589
9.388	SwaptionVolatilityStructure Class Reference	590
9.389	Sydney Class Reference	591
9.390	SymmetricSchurDecomposition Class Reference	592
9.391	TARGET Class Reference	593
9.392	TermStructure Class Reference	594
9.393	TermStructureConsistentModel Class Reference	596
9.394	TermStructureFittingParameter Class Reference	597
9.395	Thirty360 Class Reference	598
9.396	Tian Class Reference	599
9.397	TimeBasket Class Reference	600
9.398	TimeGrid Class Reference	601
9.399	Tokyo Class Reference	602
9.400	Toronto Class Reference	604
9.401	TrapezoidIntegral Class Reference	605
9.402	Tree Class Reference	607
9.403	TreeCapFloor Class Reference	608
9.404	TreeSwaption Class Reference	609
9.405	TridiagonalOperator Class Reference	610
9.406	TridiagonalOperator::TimeSetter Class Reference	612
9.407	Trigeorgis Class Reference	613
9.408	TrinomialBranching Class Reference	614

9.409	TrinomialTree Class Reference	615
9.410	TwoFactorModel Class Reference	616
9.411	TwoFactorModel::ShortRateDynamics Class Reference	617
9.412	TwoFactorModel::ShortRateTree Class Reference	618
9.413	TypePayoff Class Reference	619
9.414	UpFrontIndexedCoupon Class Reference	620
9.415	USDLibor Class Reference	621
9.416	Value Class Reference	622
9.417	VanillaEngine Class Reference	623
9.418	VanillaOption Class Reference	624
9.419	Vasicek Class Reference	626
9.420	Vasicek::Dynamics Class Reference	627
9.421	Visitor Class Template Reference	628
9.422	Warsaw Class Reference	629
9.423	Wellington Class Reference	630
9.424	Xibor Class Reference	631
9.425	XiborManager Class Reference	633
9.426	ZARLibor Class Reference	634
9.427	ZeroCurve Class Reference	635
9.428	ZeroSpreadedTermStructure Class Reference	636
9.429	ZeroYieldStructure Class Reference	638
9.430	Zurich Class Reference	640
10	QuantLib File Documentation	641
10.1	ql/argsandresults.hpp File Reference	641
10.2	ql/calendar.hpp File Reference	642
10.3	ql/Calendars/budapest.hpp File Reference	643
10.4	ql/Calendars/copenhagen.hpp File Reference	644
10.5	ql/Calendars/frankfurt.hpp File Reference	645
10.6	ql/Calendars/helsinki.hpp File Reference	646
10.7	ql/Calendars/johannesburg.hpp File Reference	647
10.8	ql/Calendars/jointcalendar.hpp File Reference	648
10.9	ql/Calendars/london.hpp File Reference	649
10.10	ql/Calendars/milan.hpp File Reference	650
10.11	ql/Calendars/newyork.hpp File Reference	651
10.12	ql/Calendars/nullcalendar.hpp File Reference	652
10.13	ql/Calendars/oslo.hpp File Reference	653

10.14	ql/Calendars/stockholm.hpp File Reference	654
10.15	ql/Calendars/sydney.hpp File Reference	655
10.16	ql/Calendars/target.hpp File Reference	656
10.17	ql/Calendars/tokyo.hpp File Reference	657
10.18	ql/Calendars/toronto.hpp File Reference	658
10.19	ql/Calendars/warsaw.hpp File Reference	659
10.20	ql/Calendars/wellington.hpp File Reference	660
10.21	ql/Calendars/zurich.hpp File Reference	661
10.22	ql/capvolstructures.hpp File Reference	662
10.23	ql/cashflow.hpp File Reference	663
10.24	ql/CashFlows/basispointsensitivity.hpp File Reference	664
10.25	ql/CashFlows/cashflowvectors.hpp File Reference	665
10.26	ql/CashFlows/coupon.hpp File Reference	666
10.27	ql/CashFlows/fixedratecoupon.hpp File Reference	667
10.28	ql/CashFlows/floatingratecoupon.hpp File Reference	668
10.29	ql/CashFlows/inarrearrindexedcoupon.hpp File Reference	669
10.30	ql/CashFlows/indexcashflowvectors.hpp File Reference	670
10.31	ql/CashFlows/indexedcoupon.hpp File Reference	671
10.32	ql/CashFlows/parcoupon.hpp File Reference	672
10.33	ql/CashFlows/shortfloatingcoupon.hpp File Reference	673
10.34	ql/CashFlows/shortindexedcoupon.hpp File Reference	674
10.35	ql/CashFlows/simplecashflow.hpp File Reference	675
10.36	ql/CashFlows/timebasket.hpp File Reference	676
10.37	ql/CashFlows/upfrontindexedcoupon.hpp File Reference	677
10.38	ql/currency.hpp File Reference	678
10.39	ql/dataformatters.hpp File Reference	679
10.40	ql/dataparsers.hpp File Reference	680
10.41	ql/date.hpp File Reference	681
10.42	ql/daycounter.hpp File Reference	682
10.43	ql/DayCounters/actual360.hpp File Reference	683
10.44	ql/DayCounters/actual365.hpp File Reference	684
10.45	ql/DayCounters/actualactual.hpp File Reference	685
10.46	ql/DayCounters/simpledaycounter.hpp File Reference	686
10.47	ql/DayCounters/thirty360.hpp File Reference	687
10.48	ql/diffusionprocess.hpp File Reference	688
10.49	ql/discretizedasset.hpp File Reference	689

10.50	ql/disposable.hpp File Reference	690
10.51	ql/errors.hpp File Reference	691
10.52	ql/exercise.hpp File Reference	693
10.53	ql/FiniteDifferences/americancondition.hpp File Reference	694
10.54	ql/FiniteDifferences/boundarycondition.hpp File Reference	695
10.55	ql/FiniteDifferences/bsmoperator.hpp File Reference	696
10.56	ql/FiniteDifferences/cranknicolson.hpp File Reference	697
10.57	ql/FiniteDifferences/dminus.hpp File Reference	698
10.58	ql/FiniteDifferences/dplus.hpp File Reference	699
10.59	ql/FiniteDifferences/dplusdminus.hpp File Reference	700
10.60	ql/FiniteDifferences/dzero.hpp File Reference	701
10.61	ql/FiniteDifferences/expliciteuler.hpp File Reference	702
10.62	ql/FiniteDifferences/fdtypedefs.hpp File Reference	703
10.63	ql/FiniteDifferences/finitedifferencemodel.hpp File Reference	704
10.64	ql/FiniteDifferences/impliciteuler.hpp File Reference	705
10.65	ql/FiniteDifferences/mixedscheme.hpp File Reference	706
10.66	ql/FiniteDifferences/onefactoroperator.hpp File Reference	707
10.67	ql/FiniteDifferences/shoutcondition.hpp File Reference	708
10.68	ql/FiniteDifferences/stepcondition.hpp File Reference	709
10.69	ql/FiniteDifferences/tridiagonaloperator.hpp File Reference	710
10.70	ql/FiniteDifferences/valueatcenter.hpp File Reference	711
10.71	ql/functions/daycounters.hpp File Reference	712
10.72	ql/functions/mathf.hpp File Reference	713
10.73	ql/functions/vols.hpp File Reference	714
10.74	ql/grid.hpp File Reference	715
10.75	ql/handle.hpp File Reference	716
10.76	ql/history.hpp File Reference	717
10.77	ql/index.hpp File Reference	718
10.78	ql/Indexes/audlibor.hpp File Reference	719
10.79	ql/Indexes/cadlibor.hpp File Reference	720
10.80	ql/Indexes/chflibor.hpp File Reference	721
10.81	ql/Indexes/euribor.hpp File Reference	722
10.82	ql/Indexes/gbplibor.hpp File Reference	723
10.83	ql/Indexes/jpylibor.hpp File Reference	724
10.84	ql/Indexes/usdlibor.hpp File Reference	725
10.85	ql/Indexes/xibor.hpp File Reference	726

10.86	ql/Indexes/xibormanager.hpp File Reference	727
10.87	ql/Indexes/zarlibor.hpp File Reference	728
10.88	ql/instrument.hpp File Reference	729
10.89	ql/Instruments/asianooption.hpp File Reference	730
10.90	ql/Instruments/barrierooption.hpp File Reference	731
10.91	ql/Instruments/basketoption.hpp File Reference	732
10.92	ql/Instruments/capfloor.hpp File Reference	733
10.93	ql/Instruments/cliquestoption.hpp File Reference	734
10.94	ql/Pricers/cliquestoption.hpp File Reference	735
10.95	ql/Instruments/forwardvanillaoption.hpp File Reference	736
10.96	ql/Instruments/multiassetoption.hpp File Reference	737
10.97	ql/Instruments/oneassetoption.hpp File Reference	738
10.98	ql/Instruments/oneassetstrikedoption.hpp File Reference	739
10.99	ql/Instruments/payoffs.hpp File Reference	740
10.100	ql/Instruments/quantoforwardvanillaoption.hpp File Reference	741
10.101	ql/Instruments/quantovanillaoption.hpp File Reference	742
10.102	ql/Instruments/simpleswap.hpp File Reference	743
10.103	ql/Instruments/stock.hpp File Reference	744
10.104	ql/Instruments/swap.hpp File Reference	745
10.105	ql/Instruments/swaption.hpp File Reference	746
10.106	ql/Instruments/vanillaoption.hpp File Reference	747
10.107	ql/Lattices/binomialtree.hpp File Reference	748
10.108	ql/Lattices/bsmlattice.hpp File Reference	749
10.109	ql/Lattices/lattice.hpp File Reference	750
10.110	ql/Lattices/lattice2d.hpp File Reference	751
10.111	ql/Lattices/tree.hpp File Reference	752
10.112	ql/Lattices/trinomialtree.hpp File Reference	753
10.113	ql/marketelement.hpp File Reference	754
10.114	ql/Math/array.hpp File Reference	755
10.115	ql/Math/beta.hpp File Reference	756
10.116	ql/Math/bicubicsplineinterpolation.hpp File Reference	757
10.117	ql/Math/bilinearinterpolation.hpp File Reference	758
10.118	ql/Math/binomialdistribution.hpp File Reference	759
10.119	ql/Math/bivariatenormaldistribution.hpp File Reference	760
10.120	ql/Math/chisquaredistribution.hpp File Reference	761
10.121	ql/Math/choleskydecomposition.hpp File Reference	762

10.122	ql/Math/comparison.hpp File Reference	763
10.123	ql/Math/cubicspline.hpp File Reference	764
10.124	ql/Math/discrepancystatistics.hpp File Reference	765
10.125	ql/Math/errorfunction.hpp File Reference	766
10.126	ql/Math/factorial.hpp File Reference	767
10.127	ql/Math/functional.hpp File Reference	768
10.128	ql/Math/gammadistribution.hpp File Reference	769
10.129	ql/Math/gaussianstatistics.hpp File Reference	770
10.130	ql/Math/generalstatistics.hpp File Reference	771
10.131	ql/Math/incompletegammma.hpp File Reference	772
10.132	ql/Math/incrementalstatistics.hpp File Reference	773
10.133	ql/Math/interpolation.hpp File Reference	774
10.134	ql/Math/interpolation2D.hpp File Reference	775
10.135	ql/Math/interpolationtraits.hpp File Reference	776
10.136	ql/Math/kronrodintegral.hpp File Reference	777
10.137	ql/Math/lexicographicalview.hpp File Reference	778
10.138	ql/Math/linearinterpolation.hpp File Reference	779
10.139	ql/Math/loglinearinterpolation.hpp File Reference	780
10.140	ql/Math/matrix.hpp File Reference	781
10.141	ql/Math/normaldistribution.hpp File Reference	782
10.142	ql/Math/poissondistribution.hpp File Reference	783
10.143	ql/Math/primenumbers.hpp File Reference	784
10.144	ql/Math/pseudosqrt.hpp File Reference	785
10.145	ql/Math/riskstatistics.hpp File Reference	786
10.146	ql/Math/segmentintegral.hpp File Reference	787
10.147	ql/Math/sequencestatistics.hpp File Reference	788
10.148	ql/Math/simpsonintegral.hpp File Reference	790
10.149	ql/Math/statistics.hpp File Reference	791
10.150	ql/Math/svd.hpp File Reference	792
10.151	ql/Math/symmetriceigenvalues.hpp File Reference	793
10.152	ql/Math/symmetricschurdecomposition.hpp File Reference	794
10.153	ql/Math/trapezoidintegral.hpp File Reference	795
10.154	ql/MonteCarlo/brownianbridge.hpp File Reference	796
10.155	ql/MonteCarlo/getcovariance.hpp File Reference	797
10.156	ql/MonteCarlo/mctraits.hpp File Reference	798
10.157	ql/MonteCarlo/mctypedefs.hpp File Reference	799

10.158	ql/MonteCarlo/montecarlomodel.hpp File Reference	800
10.159	ql/MonteCarlo/multipath.hpp File Reference	801
10.160	ql/MonteCarlo/multipathgenerator.hpp File Reference	802
10.161	ql/MonteCarlo/path.hpp File Reference	803
10.162	ql/MonteCarlo/pathgenerator.hpp File Reference	804
10.163	ql/MonteCarlo/pathpricer.hpp File Reference	805
10.164	ql/MonteCarlo/sample.hpp File Reference	806
10.165	ql/null.hpp File Reference	807
10.166	ql/numericalmethod.hpp File Reference	808
10.167	ql/Optimization/armijo.hpp File Reference	809
10.168	ql/Optimization/conjugategradient.hpp File Reference	810
10.169	ql/Optimization/constraint.hpp File Reference	811
10.170	ql/Optimization/costfunction.hpp File Reference	812
10.171	ql/Optimization/criteria.hpp File Reference	813
10.172	ql/Optimization/leastsquare.hpp File Reference	814
10.173	ql/Optimization/linesearch.hpp File Reference	815
10.174	ql/Optimization/method.hpp File Reference	816
10.175	ql/Optimization/problem.hpp File Reference	817
10.176	ql/Optimization/simplex.hpp File Reference	818
10.177	ql/Optimization/steepestdescent.hpp File Reference	819
10.178	ql/option.hpp File Reference	820
10.179	ql/Patterns/bridge.hpp File Reference	821
10.180	ql/Patterns/composite.hpp File Reference	822
10.181	ql/Patterns/curiouslyrecurring.hpp File Reference	823
10.182	ql/Patterns/lazyobject.hpp File Reference	824
10.183	ql/Patterns/observable.hpp File Reference	825
10.184	ql/Patterns/visitor.hpp File Reference	826
10.185	ql/payoff.hpp File Reference	827
10.186	ql/Pricers/continuousgeometricapo.hpp File Reference	828
10.187	ql/Pricers/discretegeometricapo.hpp File Reference	829
10.188	ql/Pricers/discretegeometricaso.hpp File Reference	830
10.189	ql/Pricers/europeanoption.hpp File Reference	831
10.190	ql/Pricers/fdamericanoption.hpp File Reference	832
10.191	ql/Pricers/fdbermudanoption.hpp File Reference	833
10.192	ql/Pricers/fdbsmoption.hpp File Reference	834
10.193	ql/Pricers/fddividendamericanoption.hpp File Reference	835

10.194	ql/Pricers/fddividendeuropeanoption.hpp File Reference	836
10.195	ql/Pricers/fddividendoption.hpp File Reference	837
10.196	ql/Pricers/fddividendshoutoption.hpp File Reference	838
10.197	ql/Pricers/fdeuropean.hpp File Reference	839
10.198	ql/Pricers/fdmultiperiodoption.hpp File Reference	840
10.199	ql/Pricers/fdshoutoption.hpp File Reference	841
10.200	ql/Pricers/fdstepconditionoption.hpp File Reference	842
10.201	ql/Pricers/mcbasket.hpp File Reference	843
10.202	ql/Pricers/mccliquetoption.hpp File Reference	844
10.203	ql/Pricers/mcdiscretearithmeticapo.hpp File Reference	845
10.204	ql/Pricers/mcdiscretearithmeticaso.hpp File Reference	846
10.205	ql/Pricers/mceverest.hpp File Reference	847
10.206	ql/Pricers/mchimalaya.hpp File Reference	848
10.207	ql/Pricers/mcmaxbasket.hpp File Reference	849
10.208	ql/Pricers/mcpagoda.hpp File Reference	850
10.209	ql/Pricers/mcperformanceoption.hpp File Reference	851
10.210	ql/Pricers/mcpricer.hpp File Reference	852
10.211	ql/Pricers/performanceoption.hpp File Reference	853
10.212	ql/Pricers/singleassetoption.hpp File Reference	854
10.213	ql/pricingengine.hpp File Reference	855
10.214	ql/PricingEngines/americanpayoffatexpiry.hpp File Reference	856
10.215	ql/PricingEngines/americanpayoffathit.hpp File Reference	857
10.216	ql/PricingEngines/Asian/analyticasianengine.hpp File Reference	858
10.217	ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference	859
10.218	ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference	860
10.219	ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference	861
10.220	ql/PricingEngines/Basket/mcbasketengine.hpp File Reference	862
10.221	ql/PricingEngines/Basket/stulzengine.hpp File Reference	863
10.222	ql/PricingEngines/blackformula.hpp File Reference	864
10.223	ql/PricingEngines/blackmodel.hpp File Reference	865
10.224	ql/PricingEngines/CapFloor/analyticalcapfloor.hpp File Reference	866
10.225	ql/PricingEngines/CapFloor/blackcapfloor.hpp File Reference	867
10.226	ql/PricingEngines/CapFloor/capfloorpricer.hpp File Reference	868
10.227	ql/PricingEngines/CapFloor/treecapfloor.hpp File Reference	869
10.228	ql/PricingEngines/Forward/forwardengine.hpp File Reference	870
10.229	ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference . . .	871

10.230	ql/PricingEngines/genericmodelengine.hpp File Reference	872
10.231	ql/PricingEngines/latticeshortratemodelengine.hpp File Reference	873
10.232	ql/PricingEngines/mcsimulation.hpp File Reference	874
10.233	ql/PricingEngines/Quanto/quantoengine.hpp File Reference	875
10.234	ql/PricingEngines/Swaption/blackswaption.hpp File Reference	876
10.235	ql/PricingEngines/Swaption/jamshidianswaption.hpp File Reference	877
10.236	ql/PricingEngines/Swaption/swaptionpricer.hpp File Reference	878
10.237	ql/PricingEngines/Swaption/treeswaption.hpp File Reference	879
10.238	ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference . .	880
10.239	ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference	881
10.240	ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference	882
10.241	ql/PricingEngines/Vanilla/binomialengine.hpp File Reference	883
10.242	ql/PricingEngines/Vanilla/bjerkhundstenglandengine.hpp File Reference	884
10.243	ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference	885
10.244	ql/PricingEngines/Vanilla/integralengine.hpp File Reference	886
10.245	ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference	887
10.246	ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference	888
10.247	ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference	889
10.248	ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference	890
10.249	ql/qldefines.hpp File Reference	891
10.250	ql/RandomNumbers/boxmullergaussianrng.hpp File Reference	893
10.251	ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference	894
10.252	ql/RandomNumbers/haltonrng.hpp File Reference	895
10.253	ql/RandomNumbers/inversecumgaussianrng.hpp File Reference	896
10.254	ql/RandomNumbers/inversecumgaussianrng.hpp File Reference	897
10.255	ql/RandomNumbers/knuthuniformrng.hpp File Reference	898
10.256	ql/RandomNumbers/lecuyeruniformrng.hpp File Reference	899
10.257	ql/RandomNumbers/mt19937uniformrng.hpp File Reference	900
10.258	ql/RandomNumbers/randomarraygenerator.hpp File Reference	901
10.259	ql/RandomNumbers/randomsequencegenerator.hpp File Reference	902
10.260	ql/RandomNumbers/rngtraits.hpp File Reference	903
10.261	ql/RandomNumbers/rngtypedefs.hpp File Reference	904
10.262	ql/RandomNumbers/sobolrng.hpp File Reference	905
10.263	ql/relinkablehandle.hpp File Reference	906
10.264	ql/scheduler.hpp File Reference	907
10.265	ql/ShortRateModels/calibrationhelper.hpp File Reference	908

10.266	ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference	909
10.267	ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference . . .	910
10.268	ql/ShortRateModels/model.hpp File Reference	911
10.269	ql/ShortRateModels/onefactormodel.hpp File Reference	912
10.270	ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference . . .	913
10.271	ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference . . .	914
10.272	ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference	915
10.273	ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference	916
10.274	ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference	917
10.275	ql/ShortRateModels/parameter.hpp File Reference	918
10.276	ql/ShortRateModels/twofactormodel.hpp File Reference	919
10.277	ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference	920
10.278	ql/solver1d.hpp File Reference	921
10.279	ql/Solvers1D/bisection.hpp File Reference	922
10.280	ql/Solvers1D/brent.hpp File Reference	923
10.281	ql/Solvers1D/falseposition.hpp File Reference	924
10.282	ql/Solvers1D/newton.hpp File Reference	925
10.283	ql/Solvers1D/newtonsafe.hpp File Reference	926
10.284	ql/Solvers1D/ridder.hpp File Reference	927
10.285	ql/Solvers1D/secant.hpp File Reference	928
10.286	ql/stochasticprocess.hpp File Reference	929
10.287	ql/swaptionvolstructure.hpp File Reference	930
10.288	ql/termstructure.hpp File Reference	931
10.289	ql/TermStructures/affinetermstructure.hpp File Reference	932
10.290	ql/TermStructures/compoundforward.hpp File Reference	933
10.291	ql/TermStructures/discountcurve.hpp File Reference	934
10.292	ql/TermStructures/drifttermstructure.hpp File Reference	935
10.293	ql/TermStructures/extendeddiscountcurve.hpp File Reference	936
10.294	ql/TermStructures/flatforward.hpp File Reference	937
10.295	ql/TermStructures/forwardspreadetermstructure.hpp File Reference	938
10.296	ql/TermStructures/IMPLIEDtermstructure.hpp File Reference	939
10.297	ql/TermStructures/piecewiseflatforward.hpp File Reference	940
10.298	ql/TermStructures/quantotermstructure.hpp File Reference	941
10.299	ql/TermStructures/ratehelpers.hpp File Reference	942
10.300	ql/TermStructures/zerocurve.hpp File Reference	943
10.301	ql/TermStructures/zerospreadetermstructure.hpp File Reference	944

10.302	ql/types.hpp File Reference	945
10.303	ql/Utilities/combiningiterator.hpp File Reference	946
10.304	ql/Utilities/couplingiterator.hpp File Reference	947
10.305	ql/Utilities/filteringiterator.hpp File Reference	948
10.306	ql/Utilities/iteratorcategories.hpp File Reference	949
10.307	ql/Utilities/processingiterator.hpp File Reference	950
10.308	ql/Utilities/steppingiterator.hpp File Reference	951
10.309	ql/Volatilities/blackconstantvol.hpp File Reference	952
10.310	ql/Volatilities/blackvariancecurve.hpp File Reference	953
10.311	ql/Volatilities/blackvariancesurface.hpp File Reference	954
10.312	ql/Volatilities/capflatvolvector.hpp File Reference	955
10.313	ql/Volatilities/impliedvoltermstructure.hpp File Reference	956
10.314	ql/Volatilities/localconstantvol.hpp File Reference	957
10.315	ql/Volatilities/localvolcurve.hpp File Reference	958
10.316	ql/Volatilities/localvolsurface.hpp File Reference	959
10.317	ql/Volatilities/swaptionvolmatrix.hpp File Reference	960
10.318	ql/voltermstructure.hpp File Reference	961
11	QuantLib Example Documentation	963
11.1	AmericanOption.cpp	963
11.2	BermudanSwaption.cpp	968
11.3	DiscreteHedging.cpp	973
11.4	EuropeanOption.cpp	978
11.5	history_iterators.cpp	986
11.6	swapvaluation.cpp	987
12	Todo List	999
13	Deprecated List	1003
14	Bug List	1005

Part I

User Manual

Chapter 1

An introduction to QuantLib

1.1 Introduction

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

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

QuantLib is free software and you are allowed to use, copy, modify, merge, publish, distribute, and/or sell copies of it under the conditions stated in the QuantLib **QuantLib License**(p. [32](#)).

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**(p. [32](#)) 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 an overview of the existing code base.

The QuantLib-users mailing list is 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.)
- Calendars: Budapest, Copenhagen, Frankfurt, Helsinki, Johannesburg, London, Milan, New York, Oslo, Stockholm, Sydney, TARGET, Tokyo, Toronto, Warsaw, Wellington, Zurich.
- NullCalendar (no holidays) for theoretical calculations.
- Joint calendars made up as holiday union of base calendars.
- Rolling conventions: Preceding, ModifiedPreceding, Following, ModifiedFollowing, MonthEndReference.
- Schedule class for date stream generation.
- Day count conventions: Actual360, Actual365, ActualActual (Bond, ISDA, AFB), 30/360 (US, European, Italian).

To do:

- Differentiate calendars depending on city+exchange, instead of city alone.

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.
- 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: Gauss-Kronrod, Simpson, trapezoid.
- 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 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, Sobol up to dimension 21,200 (8,129,334 if you really want) with unit and Jäkel initialization numbers.
- Randomized quasi-random sequences (in progress)
- 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)

To do:

- Add Faure low-discrepancy sequence.
- Randomized low-discrepancy sequences.

Patterns

- Bridge, composite, lazy object, observer/observable, 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.
- Adapt to the new pricing engine framework.
- 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.
- Evaluate proposed templization
- 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:

- Use StochasticProcess class instead of DiffusionProcess.
- 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 options with payoff at expiry.
- Analytic formula for American 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 European, Barone-Adesi and Whaley approximation for American, binomial (Cox-Ross-Rubinstein, Jarrow-Rudd, additive equiprobabilities, Trigeorgis, Tian, Leisen-Reimer), Bjerkstrand and Stensland approximation for American, integral European, Merton 76 jump-diffusion, Monte Carlo digital, Monte Carlo European.
- Engines for options described by "barrier" set of parameters: analytic down/up in/out, Monte Carlo down/up in/out
- Engines for options described by "discrete Asian" set of parameters: analytic discrete geometric average price
- Forward and forward-performance compound engines.
- Quanto compound engine.
- 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:

- 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 dependant options allow for evaluation with collected past observations.
- Define dividendRho for discrete dividends.
- Improve and test compound engines: quanto, forward, etc.

Pricers

- Cliquet option
- Analytic continuous geometric average-price option (European exercise).
- Analytic discrete geometric average-price option (European exercise).
- Analytic discrete geometric average-strike option (European exercise).
- Finite difference American option.
- Bermudan option.
- Finite difference American option with discrete dividends (buggy).
- Finite difference European option with discrete dividends.
- Finite difference Shout option with discrete dividends (buggy).
- Finite difference European option (Black-Scholes).
- Monte Carlo basket option.
- 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.
- Analytic 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.
- 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).
- Bond.

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
- Do not require today's date, only the settlement date (when discount==1.0) is necessary
- Assume no zero/forward implicit convention on input, specify them for output

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:

- Introduce BET and double BET.
- Valuation of credit default swap.
- Bootstrapping of default probability from bond, CDS, etc.

Test suite

More than 100 automated tests covering:

- American options.
- Asian options.
- Barrier options.
- Basket options (Stulz two-assets, etc.)
- Calendars.
- Caps/floors (implied volatility, etc.)
- Covariance matrices.
- Dates.
- Day counters.
- Digital options (Monte Carlo American cash-at-hit, etc.)
- Distributions.
- European options (implied volatility, etc.)
- Factorials, gamma, Poisson distribution.
- Integration algorithms.

- Cubic spline interpolations.
- Jump-diffusion engine.
- Low-discrepancy sequences.
- Quotes.
- Matrices (eigenvalues, eigenvectors, pseudo-square root, singular value decomposition).
- Mersenne-twister random-number generator.
- Finite-difference operators.
- Piecewise flat-forward term structure.
- Risk statistics.
- 1-dimensional solvers.
- Statistics.
- Swaps.
- Swaptions.
- Term structures.
- A number of old-style (to be deprecated) pricers.

To do:

- Add tests for quanto and forward engines.
- Add covariance/correlation test for SequenceStatistics
- Add single factor calibration and bermudan pricing tests
- Increase coverage.
- Move from cppUnit to boost unit-test framework.

Miscellanea

- Index classes for handling of fixed-income libor indexes (fixings, forecasting, etc.): AUD, CAD, CHF, EUR, GBP, JPY, USD, ZAR, generic.
- Cash-flow class.
- Currency enumeration.
- 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.

To do:

- IRR, duration, convexity, etc. for a sequence of cash flows
- Currency as class with methods returning related info
- Implement currency as per OMG definition
- Implement round as per OMG enumeration/definition

Documentation

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

To do:

- Add a "Getting started" page to the site
- Enable forums?
- Create a FAQ

Distribution

- Windows installer generated with NSIS.
- Included in the Debian distribution.
- RPMs available.

To do:

- Fink package

1.3 Where to get QuantLib

1.3.1 QuantLib releases

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

1.3.2 Current CVS snapshot

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

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

1.4 Installation

1.4.1 Linux/Unix/Mac OS X/Cygwin

A tarball of the source distribution is available from

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

After uncompressing the sources:

1. 'cd' to the QuantLib directory and type './configure' to configure the package for your system. Before doing this, you might want to type './configure --help' to obtain information on additional features you can enable.
2. Type 'make' to compile the package.
3. Type 'make install' to install the library. This might require administrative privileges.

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

1.4.2 Win32

An installer for the source distribution is available at

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

Visual C++ 6.0 projects files and Borland C++ makefiles are supplied for building the library. You are strongly suggested to also download and build cppUnit (<http://cppunit.sourceforge.net>) in order to build and run the QuantLib test-suite.

Before compiling the library, you might want to edit the file "ql/userconfig.hpp" and enable additional features by uncommenting the relevant lines.

1.5 Usage

To use QuantLib classes in your own code just add

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

at the beginning of your source/header files. 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 makefiles for gcc, the Borland free compiler, and Microsoft Visual C++. For the latter project files are also available.

1.5.1 Microsoft Visual C++

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

1. 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, in particular with regard to the settings under project settings, "C/C++" tab, "Code Generation". You'll have to check the "Use RTTI" option under the "C++ Language" category, too. Finally, you have to define the NOMINMAX macro.

1.6 Version history

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 31st, 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.html>)

VOLATILITY FRAMEWORK

- added Black and local volatility interface

PRICING ENGINES FRAMEWORK

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

YIELD TERM STRUCTURE

- interface revisited
- added discrete time forward methods
- added `DiscountCurve` (loglinear interpolated) and `CompoundForward` term structures
- `ForwardSpreadedTermStructure` moved under `QuantLib::TermStructures` 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.html>) and partially enforced
 - full support for Microsoft Visual Studio
 - full support for Linux/gcc
 - momentarily broken support for Metrowerks CodeWarrior
 - autoconfiscation (with specialized config*.hpp files for platforms without automake/autoconf support)
 - Include files moved under Include/ql folder and referenced as "ql/header.hpp"
 - Implemented expression templates techniques for array algebra optimization
 - Added custom iterators
 - Improved term structure
 - Added Asian, Bermudan, Shout, Cliquet, Himalaya, and Barrier options (all with greeks calculation, control variated where possible)
 - Added Helsinki and Wellington calendars
 - Improved Normal distribution related functions: cumulative, inverse cumulative, etc.
 - Added uniform and Gaussian random number generators
 - Added Statistics class (mean, variance, skewness, downside variance, etc.)

- Added RiskMeasures class: VAR, average shortfall, expected shortfall, etc.
 - Added RiskStatistics class combining Statistics and RiskMeasures
 - Added sample accumulator for multivariate analysis
 - Added Monte Carlo tools
 - Added matrix-related functions (square root, symmetric Schur decomposition)
 - Added interpolation framework (linear and cubic spline interpolation implemented).
- Installation facilities:
 - Added Win32 GUI installer for binaries
- Documentation:
 - support for Doxygen 1.2.7
 - Added man documentation

Release 0.1.1 - November 21st, 2000

Initial release.

1.7 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 (<http://quantlib.org/maillinglists.html>);
- the QuantLib FAQ (<http://quantlib.org/FAQ.html>);
- the QuantLib programming style guidelines (<http://quantlib.org/style.html>);
- a link to the QuantLib project page on SourceForge.net (<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.html>) 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.

1.8 The QuantLib Group

1.8.1 Authors

The QuantLib Group members are:

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

1.8.2 Contributors

We gratefully acknowledge contributions from Mario Aleppo, Toyin Akin, James Battle, Christopher Baus, Thomas Becker, Adolfo Benin, David Binderman, Jon Davidson, Matteo Gallivanoni, Roman Gitlin, Tomoya Kawanishi, Gilbert Peffer, Peter Schmitteckert, Enrico Sirola, Maxim Sokolov, David Schwartz, and Bernd Johannes Wuebben.

1.9 QuantLib License

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.

1.9.1 Comments on Copyright and License

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

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

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

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

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

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

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

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

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

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

Chapter 2

QuantLib components

2.1 Core classes

The base directory `ql` of the QuantLib sources contains what can be considered the QuantLib foundation, i.e., the core of concrete and abstract classes upon which the rest of the library is built.

Subfolders reflect the organization of classes and functions into definite *packages* or *class families*: the former being groups of classes forming a framework for particular applications, e.g., the PDE package for partial differential equations; and the latter being sets of classes which inherit from a common base class or otherwise share a common functionality and interface. These are the subject of later sections.

The QuantLib core classes include:

- numeric types such as `Time`, `Rate`, `Spread`, or `DiscountFactor`;
- concrete data classes such as `Date`, `Handle`, `Currency`, or `History`; base classes such as `Calendar` or `DayCounter`; and helper classes such as `Scheduler`;
- abstract base classes providing interfaces to the rest of the library. Among these, the most notable include `Instrument`, `Quote`, `TermStructure`, `Index`, `PricingEngine`, `StochasticProcess`, and others.

2.2 Date and time calculations

2.2.1 Dates

The concrete class **QuantLib::Date**(p. 237) implements the concept of date. Its functionalities include:

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

2.2.2 Calendars

The abstract class **QuantLib::Calendar**(p. 184) provides the interface for determining whether a date is a business day or a holiday for a given market, or incrementing/decrementing a date of a given number of business days. A number of calendars is contained in the `ql/Calendars` directory.

2.2.3 Day counters

The abstract class **QuantLib::DayCounter**(p. 240) 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, namely,

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

2.3 Lattice methods

The framework (corresponding to the `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`(p. 386), 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`(p. 607), classes which define the branching between nodes and transition probabilities.

2.3.1 Binomial trees

The binomial method is the simplest numerical method that can be used to price path-independent derivatives. It is usually the preferred lattice method under the Black-Scholes-Merton model. As an example, let's see the framework implemented in the `bsmlattice.hpp`(p. 749) file. It is a method based on a binomial tree, with constant short-rate (discounting). There are several approaches to build the underlying binomial tree, like Jarrow-Rudd or Cox-Ross-Rubinstein.

2.3.2 Trinomial trees

When the underlying stochastic process has a mean-reverting pattern, it is usually better to use a trinomial tree instead of a binomial tree. An example is implemented in the `QuantLib::Trinomial-Tree`(p. 615) 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

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

2.3.3 Bidimensional lattices

To come...

2.3.4 The QuantLib::DiscretizedAsset class

This class is a representation of the price of a derivative at a specific time. It is roughly an array of values, each value being associated to a state of the underlying stochastic variables. For the moment, it is only used when working with trees, but it should be quite easy to make a use of it in finite-differences methods. The two main points, when deriving classes from **QuantLib::DiscretizedAsset**([p. 260](#)), 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**.

2.4 The finite differences framework

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**(p. 302) class acts as a glue for such two steps—which are outlined in the following sections—and provides the interface of the resulting finite difference model for the end user. Furthermore, it provides the possibility of checking and operating on the solution array at each step—which is typically used to apply an exercise condition for an option. This is also outlined in a section below.

2.4.1 Differential operators

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

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

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

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

and implemented in class **QuantLib::DPlus**(p. 267); the operator D_- , defined as

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

and implemented in class **QuantLib::DMinus**(p. 265); 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**(p. 271). 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**(p. 268). 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.

2.4.2 Time schemes

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

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

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

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

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

hence

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

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

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

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

hence

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

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

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

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

hence

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

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

Each of the above evolvers forces a set of interface requirements upon the differential operator which are detailed in the documentation of the corresponding class, namely, `QuantLib::ExplicitEuler`(p. 284), `QuantLib::ImplicitEuler`(p. 354), and `QuantLib::CrankNicolson`(p. 228), respectively.

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

However, it must implement the following interface:

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

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

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

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

2.4.3 Step conditions

A finite difference model can be passed a step condition to be applied at each step during the rollback of the solution (e.g. the early exercise American condition). Such condition must be embodied in a class derived from **QuantLib::StepCondition**(p. 570) and must implement the interface of the latter, namely,

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

2.4.4 An example of finite difference model

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

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

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

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

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

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

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**(p. 302) class. The current value of the option is calculated by rolling back the solution to the current time, i.e., $t = 0$, and by taking the value corresponding at the current underlying price—which by construction corresponds to the central value provided that the number of grid points is odd.

```
unsigned int timeSteps = 365;

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

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

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

2.5 The Monte Carlo framework

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

Warning: this section of the documentation is currently outdated.

This framework (corresponding to the `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**(p. 448) class acts as a glue for such three steps — which are outlined in the following sections — and provides the interface of the resulting Monte Carlo model to the end user.

2.5.1 Path generation

The Black-Scholes equation

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

where r is the risk-free rate, σ is the volatility of the underlying, and $v = r - \sigma^2/2$, has the form of a diffusion process. According to this heuristic interpretation (1)(p. 47), paths followed by the logarithm of the underlying would be Brownian random walks with a constant drift v per unit time and a standard deviation $\sigma\sqrt{T}$ over a time T .

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

with average $\nu\Delta T_i$ 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**(p. 501) class stores the variation vector decomposed in its drift (determined) and diffusion (random) components. As shown below, this allows the implementation of anti-thetic variance reduction techniques.

The **QuantLib::MultiPath**(p. 455) 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**(p. 502) and **QuantLib::MultiPathGenerator**(p. 456).

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

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

2.5.2 Pricing an instrument on a path

The **QuantLib::PathPricer**(p. 504) 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.

2.5.3 Accumulating and averaging samples

The class **QuantLib::MonteCarloModel**(p. 448) 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**(p. 438) class is also available which wraps the typical usage of a Monte Carlo model.

Details on the Monte Carlo Pricer interface will be available in the **Pricers**(p. 49) section.

2.5.4 Examples of Monte Carlo models

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

Given a current underlying price u_0 and a path $p = [p_1, \dots, p_N]$ 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.

2.5.5 Notes

(1)(p. 42) 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 $\sigma\sqrt{T}$.

2.6 The short-rate modelling framework

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` (p. 112) class), analytical formulas for discount bonds and discount bond options are given (useful for calibration).

2.6.1 Single-factor models

The Hull & White model

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

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

The Black-Karasinski model

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

No analytical tractability here.

The extended Cox-Ingersoll-Ross model

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

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

2.6.2 Calibration

The class `CalibrationHelper` is a base class that facilitates the 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.

2.6.3 Two-factor models

2.6.4 Pricers

Analytical pricers

If the model is affine, i.e. discount bond options formulas exist, caps are easily priced since they are a portfolio of discount bond options. Such a pricer is implemented in **QuantLib::AnalyticalCapFloor**(p. 119). In the case of single-factor affine models, swaptions can be priced using the Jamshidian decomposition, implemented in **QuantLib::JamshidianSwaption**(p. 379).

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**(p. 487) class.

Using Trees

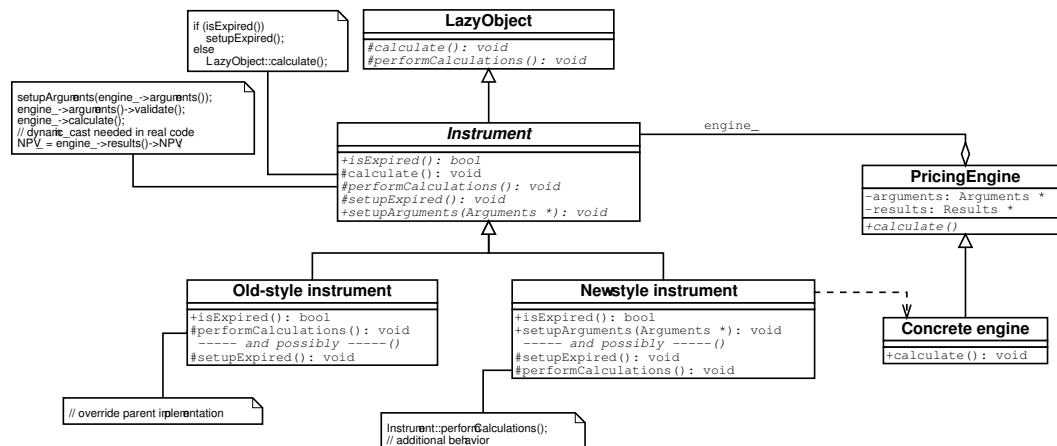
Each model derived from the single-factor model class has the ability to return a trinomial tree. For yield-curve consistent models, the fitting parameter can be determined either analytically (when possible) or numerically. When a tree is built, it is then pretty straightforward to implement a pricer for any path-independent derivative. Just implement a class derived from **NumericalDerivative** (see **QuantLib::NumericalSwaption** for example) and roll it back until the present time... Just look at **QuantLib::TreeCapFloor**(p. 608) and **QuantLib::TreeSwaption**(p. 609) for working pricers.

2.7 Currencies and FX rates

2.8 Instruments and pricers

2.8.1 Instruments and pricers

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



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

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

```

class FooArguments : public Arguments { ... };

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

class FooOptionArguments : public FooArguments { ... };

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

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

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

1. implement the `isExpired` method, and completely override the `performCalculations` method so that it bypasses the pricing-engine machinery. If the class declared any other

results beside `NPV_` and `errorEstimate_`, the `setupExpired` method should also be extended so that those results are set to a value suitable for an expired instrument. This was the migration path taken for all instruments not previously deriving from the `Option` class.

2. define suitable argument and result classes for the instrument and implement the `isExpired` and `setupArguments` methods, reusing the pricing-engine machinery provided by the default `performCalculations` method. The latter can be extended by first calling the default implementation and then performing any additional tasks required by the instrument—most often, copying additional results from the pricing engine results to the corresponding data members of the instrument. As in the previous case, the `setupExpired` method can be extended to account for such extra data members.

2.9 Math tools

Math facilities of the library include:

2.9.1 Pseudo-random number and low-discrepancy sequence generators

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

2.9.2 One-dimensional solvers

The abstract class `QuantLib::Solver1D` (p. 565) 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.

2.9.3 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` (p. 219) base class (if the gradient is not analytically implemented, it will be computed numerically).

The simplex method

This method, implemented in `QuantLib::Simplex` (p. 560), is rather raw and requires quite a lot of computing resources, but it has the advantage that it does not need any evaluation of the cost function's gradient, and that it is quite easily implemented. First, we must choose $N+1$ starting points, given here by a starting point \mathbf{P}_0 and N points such that

$$\mathbf{P}_i = \mathbf{P}_0 + \lambda \mathbf{e}_i,$$

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

The conjugate gradient method

We'll now continue with a bit more sophisticated method, implemented in `QuantLib::ConjugateGradient` (p. 213). At each step, we minimize (using Armijo's line search algorithm, implemented in `QuantLib::ArmijoLineSearch` (p. 125)) the function along a line defined by

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

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

As we can see, this optimization method requires the knowledge of the gradient of the cost function. See `QuantLib::ConjugateGradient`(p. [213](#)) .

2.10 Design patterns

2.11 Term structures

The abstract class `QuantLib::TermStructure`([p. 594](#)) provides the common interface to concrete term structure models. Among others, methods are declared which return instantaneous forward rate, discount factor, and zero rate at a given date. Adapter classes are provided which already implement part of the required methods, thus allowing the programmer to define only the non-redundant part. The `PiecewiseConstantForwards` class is provided as an example in the `ql/TermStructures` directory.

2.12 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**(p. 342), 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(p. 220) 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 $[x_0, x_m, x_{2m}, \dots]$

Chapter 3

Examples

A few examples of using the QuantLib library can be found in [chapter 11](#) of the reference manual.

Part II

Reference Manual

Chapter 4

QuantLib Module Index

4.1 QuantLib Modules

Here is a list of all modules:

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

Chapter 5

QuantLib Hierarchical Index

5.1 QuantLib Class Hierarchy

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

AcyclicVisitor	110
BPSBasketCalculator	176
BPSCalculator	177
AmericanPayoffAtExpiry	117
AmericanPayoffAtHit	118
Arguments	124
CapFloor::arguments	197
Option::arguments	491
MultiAssetOption::arguments	453
BasketOption::arguments	142
OneAssetOption::arguments	479
Swaption::arguments	587
SimpleSwap::arguments	558
Swaption::arguments	587
Array	126
ArrayFormatter	128
Average	132
Barrier	134
BarrierOption::arguments	138
BinomialDistribution	148
BivariateCumulativeNormalDistribution	152
BlackKarasinski::Dynamics	158
BoundaryCondition	172
BoundaryCondition< TridiagonalOperator >	172
DirichletBC	248
NeumannBC	460
BoxMullerGaussianRng	175
Bridge	179
Bridge< Calendar, CalendarImpl >	179
Calendar	184
Budapest	182
Copenhagen	218

Frankfurt	317
Helsinki	341
Johannesburg	381
JointCalendar	382
London	415
Milan	444
NewYork	463
NullCalendar	468
Oslo	495
Stockholm	575
Sydney	591
TARGET	593
Tokyo	602
Toronto	604
Warsaw	629
Wellington	630
Zurich	640
Bridge< Constraint, ConstraintImpl >	179
Constraint	215
BoundaryConstraint	174
CompositeConstraint	211
NoConstraint	464
PositiveConstraint	515
Bridge< DayCounter, DayCounterImpl >	179
DayCounter	240
Actual360	107
Actual365	108
ActualActual	109
SimpleDayCounter	554
Thirty360	598
Bridge< Interpolation, InterpolationImpl >	179
Interpolation	372
CubicSpline	230
MonotonicCubicSpline	447
NaturalCubicSpline	458
NaturalMonotonicCubicSpline	459
LinearInterpolation	401
LogLinearInterpolation	414
Bridge< Interpolation2D, Interpolation2DImpl >	179
Interpolation2D	373
BicubicSpline	144
BilinearInterpolation	146
Bridge< Parameter, ParameterImpl >	179
Parameter	497
ConstantParameter	214
NullParameter	469
PiecewiseConstantParameter	510
TermStructureFittingParameter	597
ExtendedCoxIngersollRoss::FittingParameter	287
G2::FittingParameter	323
HullWhite::FittingParameter	349

BrownianBridge	180
CalendarImpl	188
Calendar::WesternImpl	187
CalibrationSet	191
CLGaussianRng	203
CliquetOption::arguments	205
CliquetOptionPricer	206
combining_iterator	208
Composite	210
ConstraintImpl	216
CostFunction	219
LeastSquareFunction	394
coupling_iterator	220
CoxIngersollRoss::Dynamics	226
ExtendedCoxIngersollRoss::Dynamics	286
Cubic	229
CumulativeBinomialDistribution	232
CumulativeNormalDistribution	233
CumulativePoissonDistribution	234
CuriouslyRecurringTemplate	235
Solver1D	565
CuriouslyRecurringTemplate< Bisection >	235
Solver1D< Bisection >	565
Bisection	151
CuriouslyRecurringTemplate< Brent >	235
Solver1D< Brent >	565
Brent	178
CuriouslyRecurringTemplate< FalsePosition >	235
Solver1D< FalsePosition >	565
FalsePosition	291
CuriouslyRecurringTemplate< Newton >	235
Solver1D< Newton >	565
Newton	461
CuriouslyRecurringTemplate< NewtonSafe >	235
Solver1D< NewtonSafe >	565
NewtonSafe	462
CuriouslyRecurringTemplate< Ridder >	235
Solver1D< Ridder >	565
Ridder	540
CuriouslyRecurringTemplate< Secant >	235
Solver1D< Secant >	565
Secant	544
CurrencyFormatter	236
Date	237
DateFormatter	239
DayCounterImpl	242
DiffusionProcess	246
BlackScholesProcess	162
OrnsteinUhlenbeckProcess	493
SquareRootProcess	567

DiscreteAveragingAsianOption::arguments	257
DiscretizedAsset	260
DiscretizedDiscountBond	262
DiscretizedOption	263
Disposable	264
DoubleFormatter	266
EndCriteria	273
Error	277
AssertionFailedError	129
IllegalArgumentError	352
IllegalResultError	353
IndexError	366
OutOfMemoryError	496
PostconditionNotSatisfiedError	516
PreconditionNotSatisfiedError	517
ErrorFunction	278
EuroFormatter	280
Exercise	283
EarlyExercise	272
AmericanExercise	116
BermudanExercise	143
EuropeanExercise	281
Factorial	290
FdBermudanOption	293
FdDividendAmericanOption	296
FdDividendShoutOption	298
filtering_iterator	301
FiniteDifferenceModel	302
ForwardOptionArguments	309
GammaFunction	324
GaussianStatistics	326
GeneralStatistics	329
GenericRiskStatistics	334
HaltonRsg	338
Handle	339
Handle< AffineModel >	339
Handle< BlackModel >	339
Handle< Impl >	339
Handle< Lattice >	339
Handle< Link< TermStructure > >	339
RelinkableHandle< TermStructure >	538
Handle< Link< Type > >	339
RelinkableHandle	538
Handle< MonteCarloModel< MultiAsset< RNG >, S > >	339
Handle< MonteCarloModel< MultiAsset_old< PseudoRandomSequence_old >, Statistics > >	339
Handle< MonteCarloModel< SingleAsset< RNG >, S > >	339
Handle< MonteCarloModel< SingleAsset_old< PseudoRandom_old >, Statistics > >	339
Handle< OneFactorAffineModel >	339
Handle< path_generator_type >	339
Handle< path_pricer_type >	339
Handle< ShortRateModel >	339

Handle< TermStructure >	339
History	342
History::const_iterator	345
History::Entry	346
HullWhite::Dynamics	348
ICGaussianRng	350
ICGaussianRsg	351
IncrementalStatistics	360
IntegerFormatter	370
Interpolation2DImpl	375
Interpolation2D::templateImpl	374
BicubicSpline::Impl	145
BilinearInterpolation::Impl	147
InterpolationImpl	377
Interpolation::templateImpl	376
InverseCumulativeNormal	378
KnuthUniformRng	384
KronrodIntegral	385
LeastSquareProblem	395
LecuyerUniformRng	396
LexicographicalView	398
Linear	400
LineSearch	402
ArmijoLineSearch	125
LogLinear	413
lowest_category_iterator	416
MakeSchedule	417
Matrix	418
McPricer	438
McPricer< MultiAsset_old< PseudoRandomSequence_old > >	438
McBasket	425
McEverest	433
McHimalaya	434
McMaxBasket	435
McPagoda	436
McPricer< SingleAsset_old< PseudoRandom_old > >	438
McCliquetOption	428
McDiscreteArithmeticAPO	430
McDiscreteArithmeticASO	431
McPerformanceOption	437
McSimulation	439
McSimulation< MultiAsset< RNG >, S >	439
MCBasketEngine	426
McSimulation< SingleAsset< RNG >, S >	439
MCBarrierEngine	423
MCVanillaEngine	441
MCDigitalEngine	429
MCEuropeanEngine	432
MersenneTwisterUniformRng	443
MixedScheme	445
CrankNicolson	228

ExplicitEuler	284
ImplicitEuler	354
MonteCarloModel	448
MonteCarloModel< MultiAsset< RNG >, S >	448
MonteCarloModel< MultiAsset_old< PseudoRandomSequence_old >, Statistics >	448
MonteCarloModel< SingleAsset< RNG >, S >	448
MonteCarloModel< SingleAsset_old< PseudoRandom_old >, Statistics >	448
MoroInverseCumulativeNormal	450
MultiPath	455
MultiPathGenerator	456
MultiPathGenerator_old	457
NonLinearLeastSquare	465
NormalDistribution	466
Null	467
NumericalMethod	470
Lattice	386
BlackScholesLattice	161
Lattice2D	388
TwoFactorModel::ShortRateTree	618
OneFactorModel::ShortRateTree	486
Observable	471
AffineModel	112
G2	321
OneFactorAffineModel	483
CoxIngersollRoss	224
ExtendedCoxIngersollRoss	285
Vasicek	626
HullWhite	347
BlackModel	159
BlackVolTermStructure	170
BlackVarianceTermStructure	168
BlackVarianceCurve	164
BlackVarianceSurface	166
ImpliedVolTermStructure	357
BlackVolatilityTermStructure	169
BlackConstantVol	155
CalibrationHelper	189
CapFlatVolatilityStructure	193
CapFlatVolatilityVector	194
CapletForwardVolatilityStructure	199
CashFlow	200
Coupon	222
FixedRateCoupon	303
FloatingRateCoupon	305
IndexedCoupon	364
InArrearIndexedCoupon	359
UpFrontIndexedCoupon	620
ParCoupon	499
ShortFloatingRateCoupon	549
SimpleCashFlow	553
Index	363

Xibor	631
AUDLibor	131
CADLibor	183
CHFLibor	202
Euribor	279
GBPLibor	328
JPYLibor	383
USDLibor	621
ZARLibor	634
LazyObject	391
Instrument	367
CapFloor	195
Cap	192
Collar	207
Floor	307
Option	490
MultiAssetOption	451
BasketOption	140
OneAssetOption	476
OneAssetStrikedOption	481
BarrierOption	136
DiscreteAveragingAsianOption	255
VanillaOption	624
ForwardVanillaOption	315
QuantoVanillaOption	530
QuantoForwardVanillaOption	524
Swaption	585
Stock	574
Swap	581
SimpleSwap	556
PiecewiseFlatForward	511
Link	404
Link< TermStructure >	404
LocalVolTermStructure	412
LocalConstantVol	406
LocalVolCurve	408
LocalVolSurface	410
PricingEngine	518
GenericEngine	332
BarrierEngine	135
AnalyticBarrierEngine	120
MCBarrierEngine	423
BasketEngine	139
MCAmericanBasketEngine	422
MCBasketEngine	426
StulzEngine	578
CliquetEngine	204
DiscreteAveragingAsianEngine	254
AnalyticDiscreteAveragingAsianEngine	122
GenericModelEngine	333
AnalyticalCapFloor	119
BlackCapFloor	154

BlackSwaption	163
JamshidianSwaption	379
LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >	389
LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >	389
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >	333
GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results >	333
GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results >	333
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swap- tion::results >	333
GenericModelEngine< ShortRateModel, CapFloor::arguments, Cap- Floor::results >	333
GenericModelEngine< ShortRateModel, Swaption::arguments, Swap- tion::results >	333
VanillaEngine	623
AnalyticDigitalAmericanEngine	121
AnalyticEuropeanEngine	123
BaroneAdesiWhaleyApproximationEngine	133
BinomialVanillaEngine	150
Bjerk Sund Stensland ApproximationEngine	153
IntegralEngine	371
MCVanillaEngine	441
GenericEngine< Arguments, Results >	332
GenericModelEngine< ShortRateModel, Arguments, Results >	333
LatticeShortRateModelEngine	389
TreeCapFloor	608
TreeSwaption	609
GenericEngine< BarrierOption::arguments, BarrierOption::results >	332
GenericEngine< BasketOption::arguments, BasketOption::results >	332
GenericEngine< CapFloor::arguments, CapFloor::results >	332
GenericEngine< CliquetOption::arguments, VanillaOption::results >	332
GenericEngine< DiscreteAveragingAsianOption::arguments, DiscreteAveraging- AsianOption::results >	332
GenericEngine< ForwardOptionArguments< ArgumentsType >, ResultsType >	332
ForwardEngine	308
ForwardPerformanceEngine	310
GenericEngine< QuantoOptionArguments< ArgumentsType >, QuantoOption- Results< ResultsType > >	332
QuantoEngine	523
GenericEngine< Swaption::arguments, Swaption::results >	332
GenericEngine< VanillaOption::arguments, VanillaOption::results >	332
Quote	532
CompositeQuote	212
DerivedQuote	245
SimpleQuote	555
RateHelper	536
DepositRateHelper	243
FraRateHelper	318
FuturesRateHelper	320
SwapRateHelper	583
ShortRateModel	550
OneFactorModel	484
BlackKarasinski	157
OneFactorAffineModel	483

TwoFactorModel	616
G2	321
StochasticProcess	573
SwaptionVolatilityStructure	590
SwaptionVolatilityMatrix	589
TermStructure	594
DiscountStructure	251
AffineTermStructure	113
DiscountCurve	249
ExtendedDiscountCurve	288
ImpliedTermStructure	355
ForwardRateStructure	311
ForwardSpreadedTermStructure	313
PiecewiseFlatForward	511
ZeroYieldStructure	638
DriftTermStructure	269
QuantoTermStructure	528
ZeroCurve	635
ZeroSpreadedTermStructure	636
TermStructureConsistentModel	596
BlackKarasinski	157
ExtendedCoxIngersollRoss	285
G2	321
HullWhite	347
Observer	473
AffineTermStructure	113
BlackConstantVol	155
BlackModel	159
BlackVarianceCurve	164
BlackVarianceSurface	166
CalibrationHelper	189
CompositeQuote	212
DerivedQuote	245
DriftTermStructure	269
ExtendedDiscountCurve	288
ForwardSpreadedTermStructure	313
GenericModelEngine	333
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >	333
GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results >	333
GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results >	333
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swap- tion::results >	333
GenericModelEngine< ShortRateModel, Arguments, Results >	333
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >	333
GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results >	333
ImpliedTermStructure	355
ImpliedVolTermStructure	357
IndexedCoupon	364
LazyObject	391
Link	404
Link< TermStructure >	404
LocalConstantVol	406
LocalVolCurve	408

LocalVolSurface	410
ParCoupon	499
QuantoTermStructure	528
RateHelper	536
ShortRateModel	550
StochasticProcess	573
Xibor	631
ZeroSpreadedTermStructure	636
OneFactorModel::ShortRateDynamics	485
OptimizationMethod	488
ConjugateGradient	213
Simplex	560
SteepestDescent	569
OptionTypeFormatter	492
ParameterImpl	498
Path	501
PathGenerator	502
PathGenerator_old	503
PathPricer	504
PathPricer< MultiPath >	504
PathPricer< Path >	504
PathPricer_old	505
Payoff	506
TypePayoff	619
StrikedTypePayoff	576
AssetOrNothingPayoff	130
CashOrNothingPayoff	201
GapPayoff	325
PercentageStrikePayoff	507
PlainVanillaPayoff	513
SuperSharePayoff	579
PerformanceOption	508
Period	509
PoissonDistribution	514
PrimeNumbers	519
Problem	520
processing_iterator	521
QuantoOptionArguments	526
QuantoOptionResults	527
RandomArrayGenerator	533
RandomSequenceGenerator	534
RateFormatter	535
Results	539
Greeks	337
MultiAssetOption::results	454
OneAssetOption::results	480
MoreGreeks	449
OneAssetOption::results	480
Value	622
CapFloor::results	198
MultiAssetOption::results	454
OneAssetOption::results	480

SimpleSwap::results	559
Swaption::results	588
SalvagingAlgorithm	541
Sample	542
Schedule	543
SegmentIntegral	545
SequenceStatistics	546
SequenceStatistics< Statistics >	546
DiscrepancyStatistics	253
Short	548
SingleAssetOption	562
DiscreteGeometricAPO	258
DiscreteGeometricASO	259
EuropeanOption	282
ContinuousGeometricAPO	217
FdDividendEuropeanOption	297
FdBsmOption	294
FdEuropean	299
FdStepConditionOption	300
FdAmericanOption	292
SobolRsg	564
StatsHolder	568
StepCondition	570
AmericanCondition	115
ShoutCondition	552
StepCondition< Array >	570
stepping_iterator	571
StringFormatter	577
SVD	580
SymmetricSchurDecomposition	592
TimeBasket	600
TimeGrid	601
TrapezoidIntegral	605
SimpsonIntegral	561
Tree	607
BinomialTree	149
EqualJumpsBinomialTree	275
CoxRossRubinstein	227
Trigeorgis	613
EqualProbabilitiesBinomialTree	276
AdditiveEQPBinomialTree	111
JarrowRudd	380
LeisenReimer	397
Tian	599
TrinomialTree	615
TridiagonalOperator	610
BSMOperator	181
DMinus	265
DPlus	267
DPlusDMinus	268
DZero	271

OneFactorOperator	487
TridiagonalOperator::TimeSetter	612
TrinomialBranching	614
TwoFactorModel::ShortRateDynamics	617
Vasicek::Dynamics	627
Visitor	628
Visitor< BlackConstantVol >	628
Visitor< BlackVarianceCurve >	628
Visitor< BlackVolTermStructure >	628
Visitor< CashFlow >	628
BPSBasketCalculator	176
BPSCalculator	177
Visitor< Coupon >	628
BPSBasketCalculator	176
BPSCalculator	177
Visitor< FixedRateCoupon >	628
BPSBasketCalculator	176
XiborManager	633

Chapter 6

QuantLib Class Index

6.1 QuantLib Class List

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

Actual360 (Actual/360 day count convention)	107
Actual365 (Actual/365 day count convention)	108
ActualActual (Actual/Actual day count)	109
AcyclicVisitor (Degenerate base class for the Acyclic Visitor pattern)	110
AdditiveEQPBinomialTree (Additive equal probabilities binomial tree)	111
AffineModel (Affine model class)	112
AffineTermStructure (Term-structure implied by an affine model)	113
AmericanCondition (American exercise condition)	115
AmericanExercise (American exercise)	116
AmericanPayoffAtExpiry	117
AmericanPayoffAtHit	118
AnalyticalCapFloor (Analytical pricer for cap/floor)	119
AnalyticBarrierEngine (Pricing engine for barrier options using analytical formulae)	120
AnalyticDigitalAmericanEngine	121
AnalyticDiscreteAveragingAsianEngine (Pricing engine for European discrete geomet- ric average Asian option)	122
AnalyticEuropeanEngine (Pricing engine for European vanilla options using analytical formulae)	123
Arguments (Base class for generic argument groups)	124
ArmijoLineSearch (Armijo line search)	125
Array (1-D array used in linear algebra)	126
ArrayFormatter (Formats arrays for output)	128
AssertionFailedError (Specialized error)	129
AssetOrNothingPayoff (Binary asset-or-nothing payoff)	130
AUDLibor (AUD Libor index, also known as SIBOR)	131
Average (Placeholder for enumerated averaging types)	132
BaroneAdesiWhaleyApproximationEngine	133
Barrier (Placeholder for enumerated barrier types)	134
BarrierEngine (Barrier engine base class)	135
BarrierOption (Barrier option on a single asset)	136
BarrierOption::arguments (Arguments for barrier option calculation)	138
BasketEngine (Basket option engine base class)	139
BasketOption (Basket option on a number of assets)	140

BasketOption::arguments (Arguments for basket option calculation)	142
BermudanExercise (Bermudan exercise)	143
BicubicSpline	144
BicubicSpline::Impl (Bicubic spline implementation)	145
BilinearInterpolation (Bilinear interpolation between discrete points)	146
BilinearInterpolation::Impl (Bilinear interpolation implementation)	147
BinomialDistribution (Binomial probability distribution function)	148
BinomialTree (Binomial tree base class)	149
BinomialVanillaEngine (Pricing engine for vanilla options using binomial trees)	150
Bisection (Bisection 1-D solver)	151
BivariateCumulativeNormalDistribution (Cumulative bivariate normal distribution function)	152
BjerkstrandStenslandApproximationEngine	153
BlackCapFloor (Cap/floor priced by means of the Black formula)	154
BlackConstantVol (Constant Black volatility, no time-strike dependence)	155
BlackKarasinski (Standard Black-Karasinski model class)	157
BlackKarasinski::Dynamics (Short-rate dynamics in the Black-Karasinski model) . . .	158
BlackModel (Black-model for vanilla interest-rate derivatives)	159
BlackScholesLattice (Simple binomial lattice approximating the Black-Scholes model) .	161
BlackScholesProcess (Black-Scholes diffusion process class)	162
BlackSwaption (Swaption priced by means of the Black formula)	163
BlackVarianceCurve (Black volatility curve modelled as variance curve)	164
BlackVarianceSurface (Black volatility surface modelled as variance surface)	166
BlackVarianceTermStructure (Black variance term structure)	168
BlackVolatilityTermStructure (Black-volatility term structure)	169
BlackVolTermStructure (Black-volatility term structure)	170
BoundaryCondition (Abstract boundary condition class for finite difference problems)	172
BoundaryConstraint (Constraint imposing all arguments to be in [low,high])	174
BoxMullerGaussianRng (Gaussian random number generator)	175
BPSBasketCalculator	176
BPSCalculator (Basis point sensitivity (BPS) calculator)	177
Brent (Brent 1-D solver)	178
Bridge (The Bridge pattern made explicit)	179
BrownianBridge (Builds Wiener process paths using Gaussian variates)	180
BSMOperator (Black-Scholes-Merton differential operator)	181
Budapest (Budapest calendar)	182
CADLibor (CAD Libor index, also known as CDOR)	183
Calendar (calendar class)	184
Calendar::WesternImpl (Partial calendar implementation)	187
CalendarImpl (Abstract base class for calendar implementations)	188
CalibrationHelper (Liquid market instrument used during calibration)	189
CalibrationSet (Set of calibration instruments)	191
Cap (Concrete cap class)	192
CapFlatVolatilityStructure (Cap/floor flat volatility structure)	193
CapFlatVolatilityVector (Cap/floor at-the-money flat volatility vector)	194
CapFloor (Base class for cap-like instruments)	195
CapFloor::arguments (Arguments for cap/floor calculation)	197
CapFloor::results (Results from cap/floor calculation)	198
CapletForwardVolatilityStructure (Caplet/floorlet forward volatility structure)	199
CashFlow (Base class for cash flows)	200
CashOrNothingPayoff (Binary cash-or-nothing payoff)	201
CHFLibor (CHF Libor index, also known as ZIBOR)	202
CLGaussianRng (Gaussian random number generator)	203
CliquetEngine (Cliquet engine base class)	204

CliquetOption::arguments (Arguments for cliquet option calculation)	205
CliquetOptionPricer (Cliquet (Ratchet) option)	206
Collar (Concrete collar class)	207
combining_iterator (Iterator mapping a function to a set of underlying sequences)	208
Composite (Composite pattern)	210
CompositeConstraint (Constraint enforcing both given sub-constraints)	211
CompositeQuote (Market element whose value depends on two other market element)	212
ConjugateGradient (Multi-dimensional Conjugate Gradient class)	213
ConstantParameter (Standard constant parameter $a(t) = a$)	214
Constraint (Base constraint class)	215
ConstraintImpl (Base class for constraint implementations)	216
ContinuousGeometricAPO (Continuous geometric average-price option (European exercise))	217
Copenhagen (Copenhagen calendar)	218
CostFunction (Cost function abstract class for optimization problem)	219
coupling_iterator (Iterator mapping a function to a pair of underlying sequences)	220
Coupon (coupon accruing over a fixed period)	222
CoxIngersollRoss (Cox-Ingersoll-Ross model class)	224
CoxIngersollRoss::Dynamics (Dynamics of the short-rate under the Cox-Ingersoll-Ross model)	226
CoxRossRubinstein (Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree)	227
CrankNicolson (Crank-Nicolson scheme for finite difference methods)	228
Cubic (Cubic-spline interpolation traits)	229
CubicSpline (Cubic spline interpolation between discrete points)	230
CumulativeBinomialDistribution (Cumulative binomial distribution function)	232
CumulativeNormalDistribution (Cumulative normal distribution function)	233
CumulativePoissonDistribution (Cumulative Poisson distribution function)	234
CuriouslyRecurringTemplate (Support for the curiously recurring template pattern)	235
CurrencyFormatter (Formats currencies for output)	236
Date (Concrete date class)	237
DateFormatter (Formats dates for output)	239
DayCounter (Day counter class)	240
DayCounterImpl (Abstract base class for day counter implementations)	242
DepositRateHelper (Deposit rate)	243
DerivedQuote (Market element whose value depends on another market element)	245
DiffusionProcess (Diffusion process class)	246
DirichletBC (Neumann boundary condition (i.e., constant value))	248
DiscountCurve (Term structure based on loglinear interpolation of discount factors)	249
DiscountStructure (Discount factor term structure)	251
DiscrepancyStatistics (Statistic tool for sequences with discrepancy calculation)	253
DiscreteAveragingAsianEngine (Discrete averaging asian engine base class)	254
DiscreteAveragingAsianOption (Asian option)	255
DiscreteAveragingAsianOption::arguments (Extra arguments for single asset asian option calculation)	257
DiscreteGeometricAPO (Discrete geometric average-price Asian option (European style))	258
DiscreteGeometricASO (Discrete geometric average-strike Asian option (European style))	259
DiscretizedAsset (Discretized asset class used by numerical methods)	260
DiscretizedDiscountBond (Useful discretized discount bond asset)	262
DiscretizedOption (Discretized option on another asset)	263
Disposable (Generic disposable object with move semantics)	264
DMinus (D_- matricial representation)	265
DoubleFormatter (Formats doubles for output)	266

DPlus (D_+ matricial representation)	267
DPlusDMinus (D_+D_- matricial representation)	268
DriftTermStructure (Drift term structure)	269
DZero (D_0 matricial representation)	271
EarlyExercise (Early-exercise base class)	272
EndCriteria (Criteria to end optimization process)	273
EqualJumpsBinomialTree (Base class for equal jumps binomial tree)	275
EqualProbabilitiesBinomialTree (Base class for equal probabilities binomial tree)	276
Error (Base error class)	277
ErrorFunction (Error function)	278
Euribor (Euribor index)	279
EuroFormatter (Formats amounts in Euro for output)	280
EuropeanExercise (European exercise)	281
EuropeanOption (Black-Scholes-Merton European option)	282
Exercise (Base exercise class)	283
ExplicitEuler (Forward Euler scheme for finite difference methods)	284
ExtendedCoxIngersollRoss (Extended Cox-Ingersoll-Ross model class)	285
ExtendedCoxIngersollRoss::Dynamics (Short-rate dynamics in the extended Cox-Ingersoll-Ross model)	286
ExtendedCoxIngersollRoss::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	287
ExtendedDiscountCurve (Term structure based on loglinear interpolation of discount factors)	288
Factorial (Factorial numbers calculator)	290
FalsePosition (False position 1-D solver)	291
FdAmericanOption (American option)	292
FdBermudanOption (Bermudan option)	293
FdBsmOption (Black-Scholes-Merton option priced numerically)	294
FdDividendAmericanOption (American option with discrete dividends)	296
FdDividendEuropeanOption (European option with dividends)	297
FdDividendShoutOption (Shout option with dividends)	298
FdEuropean (Example of European option calculated using finite differences)	299
FdStepConditionOption (option executing additional code at each time step)	300
filtering_iterator (Iterator filtering undesired data)	301
FiniteDifferenceModel (Generic finite difference model)	302
FixedRateCoupon (Coupon paying a fixed interest rate)	303
FloatingRateCoupon (Coupon at par on a term structure)	305
Floor (Concrete floor class)	307
ForwardEngine (Forward engine base class)	308
ForwardOptionArguments (Arguments for forward (strike-resetting) option calculation)	309
ForwardPerformanceEngine (Forward performance engine)	310
ForwardRateStructure (Forward rate term structure)	311
ForwardSpreadedTermStructure (Term structure with added spread on the instantaneous forward rate)	313
ForwardVanillaOption (Forward version of a vanilla option)	315
Frankfurt (Frankfurt calendar)	317
FraRateHelper (Forward rate agreement)	318
FuturesRateHelper (Interest-rate futures)	320
G2 (Two-additive-factor gaussian model class)	321
G2::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	323
GammaFunction (Gamma function class)	324
GapPayoff (Binary gap payoff)	325
GaussianStatistics (Statistics tool for gaussian-assumption risk measures)	326
GBPLibor (GBP Libor index)	328

GeneralStatistics (Statistics tool)	329
GenericEngine (Template base class for option pricing engines)	332
GenericModelEngine (Base class for some pricing engine on a particular model)	333
GenericRiskStatistics (Empirical-distribution risk measures)	334
Greeks (Additional option results)	337
HaltonRsg (Halton low-discrepancy sequence generator)	338
Handle (Reference-counted pointer)	339
Helsinki (Helsinki calendar)	341
History (Container for historical data)	342
History::const_iterator (Random access iterator on history entries)	345
History::Entry (Single datum in history)	346
HullWhite (Single-factor Hull-White (extended Vasicek) model class)	347
HullWhite::Dynamics (Short-rate dynamics in the Hull-White model)	348
HullWhite::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	349
ICGaussianRng (Inverse cumulative Gaussian random number generator)	350
ICGaussianRsg (Inverse cumulative Gaussian random sequence generator)	351
IllegalArgumentError (Specialized error)	352
IllegalResultError (Specialized error)	353
ImplicitEuler (Backward Euler scheme for finite difference methods)	354
ImpliedTermStructure (Implied term structure at a given date in the future)	355
ImpliedVolTermStructure (Implied vol term structure at a given date in the future)	357
InArrearIndexedCoupon (In-arrear indexed coupon class)	359
IncrementalStatistics (Statistics tool based on incremental accumulation)	360
Index (Purely virtual base class for indexes)	363
IndexedCoupon (Base indexed coupon class)	364
IndexError (Specialized error)	366
Instrument (Abstract instrument class)	367
IntegerFormatter (Formats integers for output)	370
IntegralEngine	371
Interpolation (Base class for 1-D interpolations)	372
Interpolation2D (Base class for 2-D interpolations)	373
Interpolation2D::templateImpl (Basic template implementation)	374
Interpolation2DImpl (Abstract base class for 2-D interpolation implementations)	375
Interpolation::templateImpl (Basic template implementation)	376
InterpolationImpl (Abstract base class for interpolation implementations)	377
InverseCumulativeNormal (Inverse cumulative normal distribution function)	378
JamshidianSwaption (Jamshidian swaption pricer)	379
JarrowRudd (Jarrow-Rudd (multiplicative) equal probabilities binomial tree)	380
Johannesburg (Johannesburg calendar)	381
JointCalendar (Joint calendar)	382
JPYLibor (JPY Libor index, also known as TIBOR)	383
KnuthUniformRng (Uniform random number generator)	384
KronrodIntegral (Integral of a 1-dimensional function using the Gauss-Kronrod method)	385
Lattice (Lattice-method base class)	386
Lattice2D (Two-dimensional lattice)	388
LatticeShortRateModelEngine (Engine for a short-rate model specialized on a lattice)	389
LazyObject (Framework for calculation on demand and result caching)	391
LeastSquareFunction (Cost function for least-square problems)	394
LeastSquareProblem (Base class for least square problem)	395
LecuyerUniformRng (Uniform random number generator)	396
LeisenReimer (Leisen & Reimer tree: multiplicative approach)	397
LexicographicalView (Lexicographical 2-D view of a contiguous set of data)	398
Linear (Linear interpolation traits)	400
LinearInterpolation (Linear interpolation between discrete points)	401

LineSearch (Base class for line search)	402
Link (Relinkable access to a Handle)	404
LocalConstantVol (Constant local volatility, no time-strike dependence)	406
LocalVolCurve (Local volatility curve derived from a Black curve)	408
LocalVolSurface (Local volatility surface derived from a Black vol surface)	410
LocalVolTermStructure (Local-volatility term structure)	412
LogLinear (Log-linear interpolation traits)	413
LogLinearInterpolation	414
London (London calendar)	415
lowest_category_iterator (Most generic of two given iterator categories)	416
MakeSchedule (Helper class)	417
Matrix (Matrix used in linear algebra)	418
MCAmericanBasketEngine (Least-square Monte Carlo engine)	422
MCBarrierEngine (Pricing engine for barrier options using Monte Carlo)	423
McBasket (Simple example of multi-factor Monte Carlo pricer)	425
MCBasketEngine (MC Pricing engine for European Baskets)	426
McCliquetOption (Simple example of Monte Carlo pricer)	428
MCDigitalEngine (Pricing engine for digital options using Monte Carlo simulation)	429
McDiscreteArithmeticAPO (Example of Monte Carlo pricer using a control variate)	430
McDiscreteArithmeticASO (Example of Monte Carlo pricer using a control variate)	431
MCEuropeanEngine (European option pricing engine using Monte Carlo simulation)	432
McEverest (Everest-type option pricer)	433
McHimalaya (Himalayan-type option pricer)	434
McMaxBasket (Simple example of multi-factor Monte Carlo pricer)	435
McPagoda (Roofed Asian option)	436
McPerformanceOption (Performance option computed using Monte Carlo simulation)	437
McPricer (Base class for Monte Carlo pricers)	438
McSimulation (Base class for Monte Carlo engines)	439
MCVanillaEngine (Pricing engine for vanilla option using Monte Carlo simulation)	441
MersenneTwisterUniformRng (Uniform random number generator)	443
Milan (Milan calendar)	444
MixedScheme (Mixed (explicit/implicit) scheme for finite difference methods)	445
MonotonicCubicSpline (Cubic spline with monotonicity constraint)	447
MonteCarloModel (General purpose Monte Carlo model for path samples)	448
MoreGreeks (More additional option results)	449
MoroInverseCumulativeNormal (Moro Inverse cumulative normal distribution class)	450
MultiAssetOption (Base class for options on multiple assets)	451
MultiAssetOption::arguments (Arguments for multi-asset option calculation)	453
MultiAssetOption::results (Results from multi-asset option calculation)	454
MultiPath (Multiple random walk)	455
MultiPathGenerator (Generates a multipath from a random number generator)	456
MultiPathGenerator_old (Generates a multipath from a random number generator)	457
NaturalCubicSpline (Cubic spline with null second derivative at end points)	458
NaturalMonotonicCubicSpline (Natural cubic spline with monotonicity constraint)	459
NeumannBC (Neumann boundary condition (i.e., constant derivative))	460
Newton (Newton 1-D solver)	461
NewtonSafe (Safe Newton 1-D solver)	462
NewYork (New York calendar)	463
NoConstraint (No constraint)	464
NonLinearLeastSquare (Non-linear least-square method)	465
NormalDistribution (Normal distribution function)	466
Null (Template class providing a null value for a given type)	467
NullCalendar (Calendar for reproducing theoretical calculations)	468
NullParameter (Parameter which is always zero $a(t) = 0$)	469

NumericalMethod (Numerical method (tree, finite-differences) base class)	470
Observable (Object that notifies its changes to a set of observables)	471
Observer (Object that gets notified when a given observable changes)	473
OneAssetOption (Base class for options on a single asset)	476
OneAssetOption::arguments (Arguments for single-asset option calculation)	479
OneAssetOption::results (Results from single-asset option calculation)	480
OneAssetStrikedOption (Base class for options on a single asset with striked payoff)	481
OneFactorAffineModel (Single-factor affine base class)	483
OneFactorModel (Single-factor short-rate model abstract class)	484
OneFactorModel::ShortRateDynamics (Base class describing the short-rate dynamics)	485
OneFactorModel::ShortRateTree (Recombining trinomial tree discretizing the state variable)	486
OneFactorOperator (Interest-rate single factor model differential operator)	487
OptimizationMethod (Abstract class for constrained optimization method)	488
Option (Base option class)	490
Option::arguments	491
OptionTypeFormatter (Formats option type for output)	492
OrnsteinUhlenbeckProcess (Ornstein-Uhlenbeck process class)	493
Oslo (Oslo calendar)	495
OutOfMemoryError (Specialized error)	496
Parameter (Base class for model arguments)	497
ParameterImpl (Base class for model parameter implementation)	498
ParCoupon (coupon at par on a term structure)	499
Path	501
PathGenerator (Generates random paths using a sequence generator)	502
PathGenerator_old (Generates random paths from a random number generator)	503
PathPricer (Base class for path pricers)	504
PathPricer_old (Base class for path pricers)	505
Payoff (Base class for option payoffs)	506
PercentageStrikePayoff (Payoff with strike expressed as percentage)	507
PerformanceOption (Performance option)	508
Period (Time period described by a number of a given time unit)	509
PiecewiseConstantParameter (Piecewise-constant parameter)	510
PiecewiseFlatForward (Piecewise flat forward term structure)	511
PlainVanillaPayoff (Plain-vanilla payoff)	513
PoissonDistribution (Normal distribution function)	514
PositiveConstraint (Constraint imposing positivity to all arguments)	515
PostconditionNotSatisfiedError (Specialized error)	516
PreconditionNotSatisfiedError (Specialized error)	517
PricingEngine (Interface for pricing engines)	518
PrimeNumbers (Prime numbers calculator)	519
Problem (Constrained optimization problem)	520
processing_iterator (Iterator mapping a unary function to an underlying sequence)	521
QuantoEngine (Quanto engine base class)	523
QuantoForwardVanillaOption (Quanto version of a forward vanilla option)	524
QuantoOptionArguments (Arguments for quanto option calculation)	526
QuantoOptionResults (Results from quanto option calculation)	527
QuantoTermStructure (Quanto term structure)	528
QuantoVanillaOption (Quanto version of a vanilla option)	530
Quote (Purely virtual base class for market observables)	532
RandomArrayGenerator (Generates random arrays using a random number generator)	533
RandomSequenceGenerator (Random sequence generator based on a pseudo-random number generator)	534
RateFormatter (Formats rates for output)	535

RateHelper (Base class for rate helpers)	536
RelinkableHandle (Globally accessible relinkable pointer)	538
Results (Base class for generic result groups)	539
Ridder (Ridder 1-D solver)	540
SalvagingAlgorithm (Algorithm used for matricial pseudo square root)	541
Sample (Weighted sample)	542
Schedule (Payment schedule)	543
Secant (Secant 1-D solver)	544
SegmentIntegral (Integral of a one-dimensional function)	545
SequenceStatistics (Statistics analysis of N-dimensional (sequence) data)	546
Short (Short indexed coupon)	548
ShortFloatingRateCoupon (Short coupon at par on a term structure)	549
ShortRateModel (Abstract short-rate model class)	550
ShoutCondition (Shout option condition)	552
SimpleCashFlow (Predetermined cash flow)	553
SimpleDayCounter (Simple day counter for reproducing theoretical calculations)	554
SimpleQuote (Market element returning a stored value)	555
SimpleSwap (Simple fixed-rate vs Libor swap)	556
SimpleSwap::arguments (Arguments for simple swap calculation)	558
SimpleSwap::results (Results from simple swap calculation)	559
Simplex (Multi-dimensional simplex class)	560
SimpsonIntegral (Integral of a one-dimensional function)	561
SingleAssetOption (Black-Scholes-Merton option)	562
SobolRsg (Sobol low-discrepancy sequence generator)	564
Solver1D (Base class for 1-D solvers)	565
SquareRootProcess (Square-root process class)	567
StatsHolder (Helper class for precomputed distributions)	568
SteepestDescent (Multi-dimensional steepest-descent class)	569
StepCondition (Condition to be applied at every time step)	570
stepping_iterator (Iterator advancing in constant steps)	571
StochasticProcess (Base stochastic process class)	573
Stock (Simple stock class)	574
Stockholm (Stockholm calendar)	575
StrikedTypePayoff (Intermediate class for payoffs based on a fixed strike)	576
StringFormatter (Formats strings as lower- or uppercase)	577
StulzEngine (Pricing engine for 2D European Baskets)	578
SuperSharePayoff (Binary supershare payoff)	579
SVD (Singular value decomposition)	580
Swap (Interest rate swap)	581
SwapRateHelper (Swap rate)	583
Swaption (Swaption class)	585
Swaption::arguments (Arguments for swaption calculation)	587
Swaption::results (Results from swaption calculation)	588
SwaptionVolatilityMatrix (At-the-money swaption-volatility matrix)	589
SwaptionVolatilityStructure (Swaption-volatility structure)	590
Sydney (Sydney calendar (New South Wales, Australia))	591
SymmetricSchurDecomposition (Symmetric threshold Jacobi algorithm)	592
TARGET (TARGET calendar)	593
TermStructure (Interest-rate term structure)	594
TermStructureConsistentModel (Term-structure consistent model class)	596
TermStructureFittingParameter (Deterministic time-dependent parameter used for yield-curve fitting)	597
Thirty360 (30/360 day count convention)	598
Tian (Tian tree: third moment matching, multiplicative approach)	599

TimeBasket (Distribution over a number of dates)	600
TimeGrid (Time grid class)	601
Tokyo (Tokyo calendar)	602
Toronto (Toronto calendar)	604
TrapezoidIntegral (Integral of a one-dimensional function)	605
Tree (Tree approximating a single-factor diffusion)	607
TreeCapFloor (Cap/floor priced on a lattice)	608
TreeSwaption (Swaption priced on a lattice)	609
TridiagonalOperator (Base implementation for tridiagonal operator)	610
TridiagonalOperator::TimeSetter (Encapsulation of time-setting logic)	612
Trigeorgis (Trigeorgis (additive equal jumps) binomial tree)	613
TrinomialBranching (Branching scheme for a trinomial node)	614
TrinomialTree (Recombining trinomial tree class)	615
TwoFactorModel (Abstract base-class for two-factor models)	616
TwoFactorModel::ShortRateDynamics (Class describing the dynamics of the two state variables)	617
TwoFactorModel::ShortRateTree (Recombining two-dimensional tree discretizing the state variable)	618
TypePayoff (Intermediate class for call/put/straddle payoffs)	619
UpFrontIndexedCoupon (up front indexed coupon class)	620
USDLibor (USD Libor index)	621
Value (Pricing results)	622
VanillaEngine (Vanilla option engine base class)	623
VanillaOption (Vanilla option (no discrete dividends, no barriers) on a single asset) . .	624
Vasicek (Vasicek model class)	626
Vasicek::Dynamics (Short-rate dynamics in the Vasicek model)	627
Visitor (Visitor for a specific class)	628
Warsaw (Warsaw calendar)	629
Wellington (Wellington calendar)	630
Xibor (Base class for libor indexes)	631
XiborManager (Global repository for libor histories)	633
ZARLibor (ZAR Libor index, also known as JIBAR)	634
ZeroCurve (Term structure based on linear interpolation of zero yields)	635
ZeroSpreadedTermStructure (Term structure with an added spread on the zero yield rate)	636
ZeroYieldStructure (Zero yield term structure)	638
Zurich (Zurich calendar)	640

Chapter 7

QuantLib File Index

7.1 QuantLib File List

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

ql/argsandresults.hpp (Base classes for generic arguments and results)	641
ql/calendar.hpp (calendar class)	642
ql/capvolstructures.hpp (Cap/Floor volatility structures)	662
ql/cashflow.hpp (Base class for cash flows)	663
ql/currency.hpp (Known currencies)	678
ql/dataformatters.hpp (Classes used to format data for output)	679
ql/dataparsers.hpp (Classes used to parse data for input)	680
ql/date.hpp (Date- and time-related classes, typedefs and enumerations)	681
ql/daycounter.hpp (Day counter class)	682
ql/diffusionprocess.hpp (Diffusion process)	688
ql/discretizedasset.hpp (Discretized asset classes)	689
ql/disposable.hpp (Generic disposable object with move semantics)	690
ql/errors.hpp (Classes and functions for error handling)	691
ql/exercise.hpp (Option exercise classes and payoff function)	693
ql/grid.hpp (Grid classes with useful constructors for trees and finite diffs)	715
ql/handle.hpp (Reference-counted pointer)	716
ql/history.hpp (History class)	717
ql/index.hpp (Purely virtual base class for indexes)	718
ql/instrument.hpp (Abstract instrument class)	729
ql/marketelement.hpp (Purely virtual base class for market observables)	754
ql/null.hpp (Null values)	807
ql/numericalmethod.hpp (Numerical method class)	808
ql/option.hpp (Base option class)	820
ql/payoff.hpp (Option payoff classes)	827
ql/pricingengine.hpp (Base class for pricing engines)	855
ql/qldefines.hpp (Global definitions and compiler switches)	891
ql/relinkablehandle.hpp (Globally accessible relinkable pointer)	906
ql/scheduler.hpp (Date scheduler)	907
ql/solver1d.hpp (Abstract 1-D solver class)	921
ql/stochasticprocess.hpp (Base stochastic process class)	929
ql/swaptionvolstructure.hpp (Swaption volatility structure)	930
ql/termstructure.hpp (Term structure)	931
ql/types.hpp (Custom types)	945

ql/voltermstructure.hpp (Volatility term structures)	961
ql/Calendars/budapest.hpp (Budapest calendar)	643
ql/Calendars/copenhagen.hpp (Copenhagen calendar)	644
ql/Calendars/frankfurt.hpp (Frankfurt calendar)	645
ql/Calendars/helsinki.hpp (Helsinki calendar)	646
ql/Calendars/johannesburg.hpp (Johannesburg calendar)	647
ql/Calendars/jointcalendar.hpp (Joint calendar)	648
ql/Calendars/london.hpp (London calendar)	649
ql/Calendars/milan.hpp (Milan calendar)	650
ql/Calendars/newyork.hpp (New York calendar)	651
ql/Calendars/nullcalendar.hpp (Calendar for reproducing theoretical calculations)	652
ql/Calendars/oslo.hpp (Oslo calendar)	653
ql/Calendars/stockholm.hpp (Stockholm calendar)	654
ql/Calendars/sydney.hpp (Sydney calendar)	655
ql/Calendars/target.hpp (TARGET calendar)	656
ql/Calendars/tokyo.hpp (Tokyo calendar)	657
ql/Calendars/toronto.hpp (Toronto calendar)	658
ql/Calendars/warsaw.hpp (Warsaw calendar)	659
ql/Calendars/wellington.hpp (Wellington calendar)	660
ql/Calendars/zurich.hpp (Zurich calendar)	661
ql/CashFlows/basispointsensitivity.hpp (Basis point sensitivity calculator)	664
ql/CashFlows/cashflowvectors.hpp (Cash flow vector builders)	665
ql/CashFlows/coupon.hpp (Coupon accruing over a fixed period)	666
ql/CashFlows/fixedratecoupon.hpp (Coupon paying a fixed annual rate)	667
ql/CashFlows/floatingratecoupon.hpp (Coupon at par on a term structure)	668
ql/CashFlows/inarrearindexedcoupon.hpp (In arrear indexed coupon)	669
ql/CashFlows/indexcashflowvectors.hpp (Index Cash flow vector builders)	670
ql/CashFlows/indexedcoupon.hpp (Indexed coupon)	671
ql/CashFlows/parcoupon.hpp (Coupon at par on a term structure)	672
ql/CashFlows/shortfloatingcoupon.hpp (Short (or long) coupon at par on a term structure)	673
ql/CashFlows/shortindexedcoupon.hpp (Short (or long) indexed coupon)	674
ql/CashFlows/simplecashflow.hpp (Predetermined cash flow)	675
ql/CashFlows/timebasket.hpp	676
ql/CashFlows/upfrontindexedcoupon.hpp (Up front indexed coupon)	677
ql/DayCounters/actual360.hpp (Act/360 day counter)	683
ql/DayCounters/actual365.hpp (Act/365 day counter)	684
ql/DayCounters/actualactual.hpp (Act/act day counters)	685
ql/DayCounters/simpledaycounter.hpp (Simple day counter for reproducing theoretical calculations)	686
ql/DayCounters/thirty360.hpp (30/360 day counters)	687
ql/FiniteDifferences/americancondition.hpp (American option exercise condition)	694
ql/FiniteDifferences/boundarycondition.hpp (Boundary conditions for differential operators)	695
ql/FiniteDifferences/bsmoperator.hpp (Differential operator for Black-Scholes-Merton equation)	696
ql/FiniteDifferences/cranknicolson.hpp (Crank-Nicolson scheme for finite difference methods)	697
ql/FiniteDifferences/dminus.hpp (D_- matricial representation)	698
ql/FiniteDifferences/dplus.hpp (D_+ matricial representation)	699
ql/FiniteDifferences/dplusdminus.hpp (D_+D_- matricial representation)	700
ql/FiniteDifferences/dzero.hpp (D_0 matricial representation)	701
ql/FiniteDifferences/expliciteuler.hpp (Explicit Euler scheme for finite difference methods)	702

ql/FiniteDifferences/ fdtypedefs.hpp (Default choices for template instantiations)	703
ql/FiniteDifferences/ finitedifferencemodel.hpp (Generic finite difference model)	704
ql/FiniteDifferences/ impliciteuler.hpp (Implicit Euler scheme for finite difference meth- ods)	705
ql/FiniteDifferences/ mixedscheme.hpp (Mixed (explicit/implicit) scheme for finite dif- ference methods)	706
ql/FiniteDifferences/ onefactoroperator.hpp (General differential operator for one-factor interest rate models)	707
ql/FiniteDifferences/ shoutcondition.hpp (Shout option exercise condition)	708
ql/FiniteDifferences/ stepcondition.hpp (Conditions to be applied at every time step) . .	709
ql/FiniteDifferences/ tridiagonaloperator.hpp (Tridiagonal operator)	710
ql/FiniteDifferences/ valueatcenter.hpp (Compute value, first, and second derivatives at grid center)	711
ql/functions/ daycounters.hpp (Day counters functions)	712
ql/functions/ mathf.hpp (Math functions)	713
ql/functions/ vols.hpp (Volatility functions)	714
ql/Indexes/ audlibor.hpp (AUD Libor index (check settlement days))	719
ql/Indexes/ cadlibor.hpp (CAD Libor index (Also known as CDOR))	720
ql/Indexes/ chflibor.hpp (CHF Libor index (Also known as ZIBOR))	721
ql/Indexes/ euribor.hpp (Euribor index)	722
ql/Indexes/ gbplibor.hpp (GBP Libor index)	723
ql/Indexes/ jpylibor.hpp (JPY Libor index (Also known as TIBOR, check settlement days))	724
ql/Indexes/ usdlibor.hpp (USD Libor index)	725
ql/Indexes/ xibor.hpp (Base class for libor indexes)	726
ql/Indexes/ xibormanager.hpp (Global repository for Xibor histories)	727
ql/Indexes/ zarlibor.hpp (ZAR Libor index (also known as JIBAR))	728
ql/Instruments/ asianoption.hpp (Asian option on a single asset)	730
ql/Instruments/ barrieroption.hpp (Barrier option on a single asset)	731
ql/Instruments/ basketoption.hpp (Basket option on a number of assets)	732
ql/Instruments/ capfloor.hpp (Cap and Floor class)	733
ql/Instruments/ cliquetoption.hpp (Cliquet option)	734
ql/Instruments/ forwardvanillaoption.hpp (Forward version of a vanilla option)	736
ql/Instruments/ multiassetoption.hpp (Option on multiple assets)	737
ql/Instruments/ oneassetoption.hpp (Option on a single asset)	738
ql/Instruments/ oneassetstrikedoption.hpp (Option on a single asset with striked payoff)	739
ql/Instruments/ payoffs.hpp (Payoffs for various options)	740
ql/Instruments/ quantoforwardvanillaoption.hpp (Quanto version of a forward vanilla option)	741
ql/Instruments/ quantovanillaoption.hpp (Quanto version of a vanilla option)	742
ql/Instruments/ simpleswap.hpp (Simple fixed-rate vs Libor swap)	743
ql/Instruments/ stock.hpp (Concrete stock class)	744
ql/Instruments/ swap.hpp (Interest rate swap)	745
ql/Instruments/ swaption.hpp (Swaption class)	746
ql/Instruments/ vanillaoption.hpp (Vanilla option on a single asset)	747
ql/Lattices/ binomialtree.hpp (Binomial tree class)	748
ql/Lattices/ bsmlattice.hpp (Binomial trees under the BSM model)	749
ql/Lattices/ lattice.hpp (Lattice method class)	750
ql/Lattices/ lattice2d.hpp (Two-dimensional tree class)	751
ql/Lattices/ tree.hpp (Tree class)	752
ql/Lattices/ trinomialtree.hpp (Trinomial tree class)	753
ql/Math/ array.hpp (1-D array used in linear algebra)	755
ql/Math/ beta.hpp (Beta and beta incomplete functions)	756
ql/Math/ bicubicsplineinterpolation.hpp (Bicubic spline interpolation between discrete points)	757

ql/Math/ bilinearinterpolation.hpp (Bilinear interpolation between discrete points) . .	758
ql/Math/ binomialdistribution.hpp (Binomial distribution)	759
ql/Math/ bivariatenormaldistribution.hpp (Bivariate cumulative normal distribution) .	760
ql/Math/ chisquaredistribution.hpp (Chi-square (central and non-central) distributions)	761
ql/Math/ choleskydecomposition.hpp (Cholesky decomposition)	762
ql/Math/ comparison.hpp (Floating-point comparisons)	763
ql/Math/ cubicspline.hpp (Cubic spline interpolation between discrete points)	764
ql/Math/ discrepancystatistics.hpp (Statistic tool for sequences with discrepancy calculation)	765
ql/Math/ errorfunction.hpp (Error function)	766
ql/Math/ factorial.hpp (Factorial numbers calculator)	767
ql/Math/ functional.hpp (Functionals and combinators not included in the STL)	768
ql/Math/ gammadistribution.hpp (Gamma distribution)	769
ql/Math/ gaussianstatistics.hpp (Statistics tool for gaussian-assumption risk measures)	770
ql/Math/ generalstatistics.hpp (Statistics tool)	771
ql/Math/ incompletegamma.hpp (Incomplete Gamma function)	772
ql/Math/ incrementalstatistics.hpp (Statistics tool based on incremental accumulation) .	773
ql/Math/ interpolation.hpp (Base class for 1-D interpolations)	774
ql/Math/ interpolation2D.hpp (Abstract base classes for 2-D interpolations)	775
ql/Math/ interpolationtraits.hpp (Traits classes for interpolation algorithms)	776
ql/Math/ kronrodintegral.hpp (Integral of a 1-dimensional function using the Gauss-Kronrod method)	777
ql/Math/ lexicographicalview.hpp (Lexicographical 2-D view of a contiguous set of data)	778
ql/Math/ linearinterpolation.hpp (Linear interpolation between discrete points)	779
ql/Math/ loglinearinterpolation.hpp (Log-linear interpolation between discrete points)	780
ql/Math/ matrix.hpp (Matrix used in linear algebra)	781
ql/Math/ normaldistribution.hpp (Normal, cumulative and inverse cumulative distributions)	782
ql/Math/ poissondistribution.hpp (Poisson distribution)	783
ql/Math/ primenumbers.hpp (Prime numbers calculator)	784
ql/Math/ pseudosqrt.hpp (Pseudo square root of a real symmetric matrix)	785
ql/Math/ riskstatistics.hpp (Empirical-distribution risk measures)	786
ql/Math/ segmentintegral.hpp (Integral of a one-dimensional function)	787
ql/Math/ sequencestatistics.hpp (Statistics tools for sequence (vector, list, array) samples)	788
ql/Math/ simpsonintegral.hpp (Integral of a one-dimensional function)	790
ql/Math/ statistics.hpp (Statistics tool with risk measures)	791
ql/Math/ svd.hpp (Singular value decomposition)	792
ql/Math/ symmetriceigenvalues.hpp (Eigenvalues / eigenvectors of a real symmetric matrix)	793
ql/Math/ symmetricschurdecomposition.hpp (Eigenvalues / eigenvectors of a real symmetric matrix)	794
ql/Math/ trapezoidintegral.hpp (Integral of a one-dimensional function)	795
ql/MonteCarlo/ brownianbridge.hpp (Brownian bridge)	796
ql/MonteCarlo/ getcovariance.hpp (Covariance matrix calculation)	797
ql/MonteCarlo/ mctrails.hpp (Monte Carlo policies)	798
ql/MonteCarlo/ mctypedefs.hpp (Default choices for template instantiations)	799
ql/MonteCarlo/ montecarlomodel.hpp (General purpose Monte Carlo model)	800
ql/MonteCarlo/ multipath.hpp (Correlated multiple asset paths)	801
ql/MonteCarlo/ multipathgenerator.hpp (Generates a multi path from a random-array generator)	802
ql/MonteCarlo/ path.hpp (Single factor random walk)	803
ql/MonteCarlo/ pathgenerator.hpp (Generates random paths using a sequence generator)	804
ql/MonteCarlo/ pathpricer.hpp (Base class for single-path pricers)	805
ql/MonteCarlo/ sample.hpp (Weighted sample)	806

ql/Optimization/armijo.hpp (Armijo line-search class)	809
ql/Optimization/conjugategradient.hpp (Conjugate gradient optimization method) . .	810
ql/Optimization/constraint.hpp (Abstract constraint class)	811
ql/Optimization/costfunction.hpp (Optimization cost function class)	812
ql/Optimization/criteria.hpp (Optimization criteria class)	813
ql/Optimization/leastsquare.hpp (Least square cost function)	814
ql/Optimization/linesearch.hpp (Line search abstract class)	815
ql/Optimization/method.hpp (Abstract optimization method class)	816
ql/Optimization/problem.hpp (Abstract optimization class)	817
ql/Optimization/simplex.hpp (Simplex optimization method)	818
ql/Optimization/steepestdescent.hpp (Steepest descent optimization method)	819
ql/Patterns/bridge.hpp (Bridge pattern (a.k.a. handle-body idiom))	821
ql/Patterns/composite.hpp (Composite pattern)	822
ql/Patterns/curiouslyrecurring.hpp (Curiously recurring template pattern)	823
ql/Patterns/lazyobject.hpp (Framework for calculation on demand and result caching) .	824
ql/Patterns/observable.hpp (Observer/observable pattern)	825
ql/Patterns/visitor.hpp (Degenerate base class for the Acyclic Visitor pattern)	826
ql/Pricers/cliquetoption.hpp (Cliquet option)	735
ql/Pricers/continuousgeometricapo.hpp (Continuous Geometric Average Price Option (European exercise))	828
ql/Pricers/discretegeometricapo.hpp (Discrete Geometric Average Price Option)	829
ql/Pricers/discretegeometricaso.hpp (Discrete Geometric Average Strike Option)	830
ql/Pricers/europeanoption.hpp (European option)	831
ql/Pricers/fdamericanoption.hpp (American option)	832
ql/Pricers/fdbermudanoption.hpp (Finite-difference evaluation of Bermudan option) .	833
ql/Pricers/fdbsmoption.hpp (Common code for numerical option evaluation)	834
ql/Pricers/fddividendamericanoption.hpp (American option with discrete deterministic dividends)	835
ql/Pricers/fddividendeuropeanoption.hpp (European option with discrete determinis- tic dividends)	836
ql/Pricers/fddividendoption.hpp (Base class for option with dividends)	837
ql/Pricers/fddividendshoutoption.hpp (Base class for shout option with dividends) . .	838
ql/Pricers/fdeuropean.hpp (Example of European option calculated using finite differ- ences)	839
ql/Pricers/fdmultiplieroption.hpp (Base class for option with events happening at different periods)	840
ql/Pricers/fdshoutoption.hpp (Shout option)	841
ql/Pricers/fdstepconditionoption.hpp (Option requiring additional code to be executed at each time step)	842
ql/Pricers/mcbbasket.hpp (Simple example of multi-factor Monte Carlo pricer)	843
ql/Pricers/mccliquetoption.hpp (Cliquet option priced with Monte Carlo simulation) .	844
ql/Pricers/mcdiscretearithmeticapo.hpp (Discrete Arithmetic Average Price Option) . .	845
ql/Pricers/mcdiscretearithmeticaso.hpp (Discrete Arithmetic Average Strike Option) .	846
ql/Pricers/mceverest.hpp (Everest-type option pricer)	847
ql/Pricers/mchimalaya.hpp (Himalayan-type option pricer)	848
ql/Pricers/mcmaxbasket.hpp (Max Basket Monte Carlo pricer)	849
ql/Pricers/mcpagoda.hpp (Roofed multi asset Asian option)	850
ql/Pricers/mcperformanceoption.hpp (Performance option priced with Monte Carlo simulation)	851
ql/Pricers/mcpricer.hpp (Base class for Monte Carlo pricers)	852
ql/Pricers/performanceoption.hpp (Performance option)	853
ql/Pricers/singleassetoption.hpp (Common code for option evaluation)	854
ql/PricingEngines/americanpayoffatexpiry.hpp (Analytical formulae for american exer- cise with payoff at expiry)	856

ql/PricingEngines/ americanpayoffathit.hpp (Analytical formulae for american exercise with payoff at hit)	857
ql/PricingEngines/ blackformula.hpp (Black formula)	864
ql/PricingEngines/ blackmodel.hpp (Abstract class for Black-type models (market models))	865
ql/PricingEngines/ genericmodelengine.hpp (Generic option engine based on a model)	872
ql/PricingEngines/ latticeshortratemodelengine.hpp (Engine for a short-rate model specialized on a lattice)	873
ql/PricingEngines/ mcsimulation.hpp (Framework for Monte Carlo engines)	874
ql/PricingEngines/Asian/ analyticasianengine.hpp (Analytic Asian option engine)	858
ql/PricingEngines/Barrier/ analyticbarrierengine.hpp (Analytic barrier option engines)	859
ql/PricingEngines/Barrier/ mcbarrierengine.hpp (Monte Carlo barrier option engines)	860
ql/PricingEngines/Basket/ mcamericanbasketengine.hpp (Least-square Monte Carlo engines)	861
ql/PricingEngines/Basket/ mcbasketengine.hpp (European Basket MC Engine)	862
ql/PricingEngines/Basket/ stulzengine.hpp (2D European Basket formulae, due to Stulz (1982))	863
ql/PricingEngines/CapFloor/ analyticalcapfloor.hpp (Analytical pricer for caps/floors)	866
ql/PricingEngines/CapFloor/ blackcapfloor.hpp (CapFloor calculated using the Black formula)	867
ql/PricingEngines/CapFloor/ capfloorpricer.hpp (Cap and floor pricer class)	868
ql/PricingEngines/CapFloor/ treecapfloor.hpp (Cap/Floor calculated using a tree)	869
ql/PricingEngines/Forward/ forwardengine.hpp (Forward (strike-resetting) option engine)	870
ql/PricingEngines/Forward/ forwardperformanceengine.hpp (Forward (strike-resetting) performance option engines)	871
ql/PricingEngines/Quanto/ quantoengine.hpp (Quanto option engine)	875
ql/PricingEngines/Swaption/ blackswaption.hpp (Swaption calculated using the Black formula)	876
ql/PricingEngines/Swaption/ jamshidianswaption.hpp (Swaption pricer using Jamshidian's decomposition)	877
ql/PricingEngines/Swaption/ swaptionpricer.hpp (Swaption pricer class)	878
ql/PricingEngines/Swaption/ treewswaption.hpp (Swaption computed using a lattice)	879
ql/PricingEngines/Vanilla/ analyticdigitalamericanengine.hpp (Analytic digital American option engine)	880
ql/PricingEngines/Vanilla/ analyticeuropeanengine.hpp (Analytic European engine)	881
ql/PricingEngines/Vanilla/ baroneadesiwhaleyengine.hpp (Barone-Adesi and Whaley approximation engine)	882
ql/PricingEngines/Vanilla/ binomialengine.hpp (Binomial option engine)	883
ql/PricingEngines/Vanilla/ bjerkhundstendslandengine.hpp (Bjerkhund and Stensland approximation engine)	884
ql/PricingEngines/Vanilla/ discretizedvanillaoption.hpp (Discretized vanilla option)	885
ql/PricingEngines/Vanilla/ integralengine.hpp (Integral option engine)	886
ql/PricingEngines/Vanilla/ jumpdiffusionengine.hpp (Jump diffusion (Merton 1976) engine)	887
ql/PricingEngines/Vanilla/ mcdigitalengine.hpp (Digital option Monte Carlo engine)	888
ql/PricingEngines/Vanilla/ mceuropeanengine.hpp (Monte Carlo European option engine)	889
ql/PricingEngines/Vanilla/ mcvanillaengine.hpp (Monte Carlo vanilla option engine)	890
ql/RandomNumbers/ boxmullergaussianrng.hpp (Box-Muller Gaussian random-number generator)	893
ql/RandomNumbers/ centrallimitgaussianrng.hpp (Central limit Gaussian random-number generator)	894
ql/RandomNumbers/ haltonrsg.hpp (Halton low-discrepancy sequence generator)	895

ql/RandomNumbers/ inversecumgaussianrng.hpp (Inverse cumulative Gaussian random-number generator)	896
ql/RandomNumbers/ inversecumgaussianrsg.hpp (Inverse cumulative Gaussian random sequence generator)	897
ql/RandomNumbers/ knuthuniformrng.hpp (Knuth uniform random number generator)	898
ql/RandomNumbers/ lecuyeruniformrng.hpp (L'Ecuyer uniform random number generator)	899
ql/RandomNumbers/ mt19937uniformrng.hpp (Mersenne Twister uniform random number generator)	900
ql/RandomNumbers/ randomarraygenerator.hpp (Generates random arrays from a random number generator)	901
ql/RandomNumbers/ randomsequencegenerator.hpp (Random sequence generator based on a pseudo-random number generator)	902
ql/RandomNumbers/ rngtraits.hpp (Random-number generation policies)	903
ql/RandomNumbers/ rngtypedefs.hpp (Default choices for template instantiations)	904
ql/RandomNumbers/ sobolrsg.hpp (Sobol low-discrepancy sequence generator)	905
ql/ShortRateModels/ calibrationhelper.hpp (Calibration helper class)	908
ql/ShortRateModels/ model.hpp (Abstract interest rate model class)	911
ql/ShortRateModels/ onefactormodel.hpp (Abstract one-factor interest rate model class)	912
ql/ShortRateModels/ parameter.hpp (Model parameter classes)	918
ql/ShortRateModels/ twofactormodel.hpp (Abstract two-factor interest rate model class)	919
ql/ShortRateModels/CalibrationHelpers/ caphelper.hpp (CapHelper calibration helper)	909
ql/ShortRateModels/CalibrationHelpers/ swaptionhelper.hpp (Swaption calibration helper)	910
ql/ShortRateModels/OneFactorModels/ blackkarasinski.hpp (Black-Karasinski model)	913
ql/ShortRateModels/OneFactorModels/ coxingersollross.hpp (Cox-Ingersoll-Ross model)	914
ql/ShortRateModels/OneFactorModels/ extendedcoxingersollross.hpp (Extended Cox-Ingersoll-Ross model)	915
ql/ShortRateModels/OneFactorModels/ hullwhite.hpp (Hull & White (HW) model)	916
ql/ShortRateModels/OneFactorModels/ vasicek.hpp (Vasicek model class)	917
ql/ShortRateModels/TwoFactorModels/ g2.hpp (Two-factor additive Gaussian Model G2++)	920
ql/Solvers1D/ bisection.hpp (Bisection 1-D solver)	922
ql/Solvers1D/ brent.hpp (Brent 1-D solver)	923
ql/Solvers1D/ falseposition.hpp (False-position 1-D solver)	924
ql/Solvers1D/ newton.hpp (Newton 1-D solver)	925
ql/Solvers1D/ newtonsafe.hpp (Safe (bracketed) Newton 1-D solver)	926
ql/Solvers1D/ ridder.hpp (Ridder 1-D solver)	927
ql/Solvers1D/ secant.hpp (Secant 1-D solver)	928
ql/TermStructures/ affinetermstructure.hpp (Affine term structure)	932
ql/TermStructures/ compoundedforward.hpp (Compounded forward term structure)	933
ql/TermStructures/ discountcurve.hpp (Pre-bootstrapped discount factor structure)	934
ql/TermStructures/ drifttermstructure.hpp (Drift term structure)	935
ql/TermStructures/ extendeddiscountcurve.hpp (Discount factor structure with detailed compound-forward calculation)	936
ql/TermStructures/ flatforward.hpp (Flat forward rate term structure)	937
ql/TermStructures/ forwardspreadedtermstructure.hpp (Forward spreaded term structure)	938
ql/TermStructures/ impliedtermstructure.hpp (Implied term structure)	939
ql/TermStructures/ piecwiseflatforward.hpp (Piecewise flat forward term structure)	940
ql/TermStructures/ quantotermstructure.hpp (Quanto term structure)	941
ql/TermStructures/ ratehelpers.hpp (Rate helpers base class)	942
ql/TermStructures/ zerocurve.hpp (Pre-bootstrapped zero curve structure)	943

ql/TermStructures/ zerospreadedtermstructure.hpp (Zero spreaded term structure) . . .	944
ql/Utilities/ combiningiterator.hpp (Iterator mapping a function to a set of underlying sequences)	946
ql/Utilities/ couplingiterator.hpp (Iterator mapping a function to a pair of underlying sequences)	947
ql/Utilities/ filteringiterator.hpp (Iterator filtering undesired data)	948
ql/Utilities/ iteratorcategories.hpp (Lowest common denominator between two iterator categories)	949
ql/Utilities/ processingiterator.hpp (Iterator mapping a unary function to an underlying sequence)	950
ql/Utilities/ steppingiterator.hpp (Iterator advancing in constant steps)	951
ql/Volatilities/ blackconstantvol.hpp (Black constant volatility, no time dependence, no strike dependence)	952
ql/Volatilities/ blackvariancecurve.hpp (Black volatility curve modelled as variance curve)	953
ql/Volatilities/ blackvariancesurface.hpp (Black volatility surface modelled as variance surface)	954
ql/Volatilities/ capflatvolvector.hpp (Cap/floor at-the-money flat volatility vector)	955
ql/Volatilities/ impliedvoltermstructure.hpp (Implied Black Vol Term Structure)	956
ql/Volatilities/ localconstantvol.hpp (Local constant volatility, no time dependence, no asset dependence)	957
ql/Volatilities/ localvolcurve.hpp (Local volatility curve derived from a Black curve) . .	958
ql/Volatilities/ localvolsurface.hpp (Local volatility surface derived from a Black vol surface)	959
ql/Volatilities/ swaptionvolmatrix.hpp (Swaption at-the-money volatility matrix)	960

Chapter 8

QuantLib Module Documentation

8.1 Global QuantLib macros

8.1.1 Detailed Description

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

Defines

- `#define QL_HEX_VERSION 0x000306f0`
version hexadecimal number
- `#define QL_VERSION "0.3.6"`
version string
- `#define QL_DUMMY_RETURN(x)`
specific per-compiler definitions Is a dummy return statement required?
- `#define QL_IO_INIT`
I/O initialization.

8.1.2 Define Documentation

8.1.2.1 `#define QL_DUMMY_RETURN(x)`

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

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

```
if (condition)
    return validResult;
else
    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.

8.1.2.2 `#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.

8.2 Math functions

8.2.1 Detailed Description

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

Defines

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

8.3 Numeric limits

8.3.1 Detailed Description

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

Defines

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

8.3.2 Define Documentation

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

Defines the value of the largest representable negative integer value

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

Defines the value of the largest representable integer value

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

Defines the value of the largest representable negative double value

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

Defines the value of the smallest representable positive double value

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

Defines the value of the largest representable double value

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

Defines the machine precision for operations over doubles

8.4 Time functions

8.4.1 Detailed Description

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

Defines

- `#define QL_CLOCK std::clock`
clock value
- `#define QL_TIME std::time`
time value

8.5 String functions

8.5.1 Detailed Description

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

Defines

- `#define QL_STRLEN std::strlen`
string length

8.6 Character functions

8.6.1 Detailed Description

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

Defines

- `#define QL_TOUPPER std::toupper`
convert to uppercase
- `#define QL_TOLOWER std::tolower`
convert to lowercase

8.7 Input/output functions

8.7.1 Detailed Description

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

Defines

- `#define QL_SPRINTF std::sprintf`
print to string

8.8 Min and max functions

8.8.1 Detailed Description

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

Defines

- `#define QL_MIN std::min`
minimum between two elements
- `#define QL_MAX std::max`
maximum between two elements

8.9 Template capabilities

8.9.1 Detailed Description

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

Defines

- **#define QL_DECLARE_TEMPLATE_SPECIALIZATIONS**
Blame Microsoft for this one...
- **#define QL_ALLOW_TEMPLATE_METHOD_CALLS 1**
Blame Microsoft for this one...
- **#define QL_TYPENAME typename**
Blame Microsoft for this one...
- **#define QL_TEMPLATE_METAPROGRAMMING_WORKS 1**

8.9.2 Define Documentation

8.9.2.1 #define QL_DECLARE_TEMPLATE_SPECIALIZATIONS

Blame Microsoft for this one...

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

8.9.2.2 #define QL_ALLOW_TEMPLATE_METHOD_CALLS 1

Blame Microsoft for this one...

Their compiler cannot cope with method calls such as

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

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

8.9.2.3 #define QL_TYPENAME typename

Blame Microsoft for this one...

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

8.9.2.4 `#define QL_TEMPLATE_METAPROGRAMMING_WORKS 1`

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

8.10 Iterator support

8.10.1 Detailed Description

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

Defines

- `#define QL_ITERATOR std::iterator`
- `#define QL_ITERATOR_TRAITS std::iterator_traits`
- `#define QL_SPECIALIZE_ITERATOR_TRAITS(T)`
- `#define QL_REVERSE_ITERATOR(iterator, type) std::reverse_iterator< iterator >`
Blame Microsoft for this one...
- `#define QL_FULL_ITERATOR_SUPPORT`

8.10.2 Define Documentation

8.10.2.1 `#define QL_ITERATOR std::iterator`

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

8.10.2.2 `#define QL_ITERATOR_TRAITS std::iterator_traits`

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

8.10.2.3 `#define QL_SPECIALIZE_ITERATOR_TRAITS(T)`

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

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

Blame Microsoft for this one...

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

8.10.2.5 `#define QL_FULL_ITERATOR_SUPPORT`

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

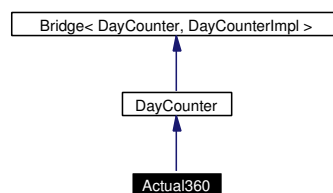
Chapter 9

QuantLib Class Documentation

9.1 Actual360 Class Reference

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

Inheritance diagram for Actual360:



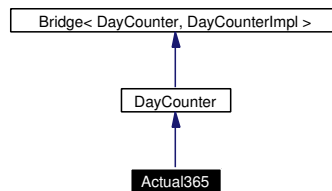
9.1.1 Detailed Description

Actual/360 day count convention.

9.2 Actual365 Class Reference

```
#include <ql/DayCounters/actual365.hpp>
```

Inheritance diagram for Actual365:



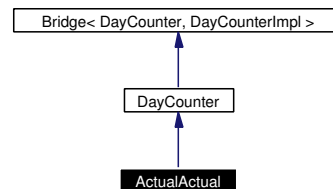
9.2.1 Detailed Description

Actual/365 day count convention.

9.3 ActualActual Class Reference

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

Inheritance diagram for ActualActual:



9.3.1 Detailed Description

Actual/Actual day count.

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

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

Public Types

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

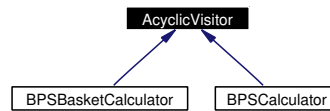
Public Member Functions

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

9.4 AcyclicVisitor Class Reference

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

Inheritance diagram for AcyclicVisitor:



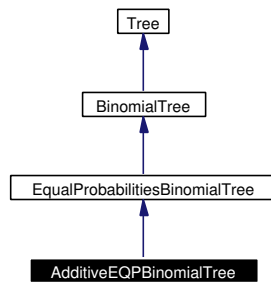
9.4.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

9.5 AdditiveEQPBinoialTree Class Reference

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

Inheritance diagram for AdditiveEQPBinoialTree:



9.5.1 Detailed Description

Additive equal probabilities binomial tree.

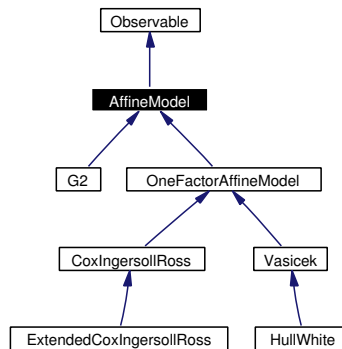
Public Member Functions

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

9.6 AffineModel Class Reference

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

Inheritance diagram for AffineModel:



9.6.1 Detailed Description

Affine model class.

This is the base class for analytically tractable models

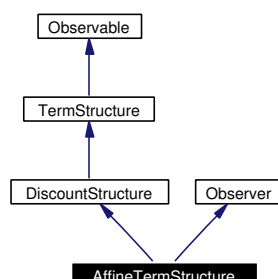
Public Member Functions

- virtual DiscountFactor **discount** (Time t) const=0
Implied discount curve.
- virtual double **discountBondOption** (Option::Type type, double strike, Time maturity, Time bondMaturity) const=0

9.7 AffineTermStructure Class Reference

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

Inheritance diagram for AffineTermStructure:



9.7.1 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](#)(p. 626) 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** &todayDate, const **Date** &referenceDate, const **Handle**<**AffineModel**> &model, const **DayCounter** &dayCounter)
constructor using a fixed model
- **AffineTermStructure** (const **Date** &todayDate, const **Date** &referenceDate, const **Handle**<**AffineModel**> &model, const std::vector< **Handle**<**RateHelper**> > &, const **Handle**<**OptimizationMethod**> &method, const **DayCounter** &dayCounter)
constructor using a model that has to be calibrated
- **DayCounter** dayCounter () const
the day counter used for date/time conversion
- **Date** todayDate () const
today's date
- **Date** referenceDate () const
the reference date, i.e., the date at which discount = 1
- **Date** maxDate () const
the latest date for which the curve can return rates
- void update ()

Protected Member Functions

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

9.7.2 Member Function Documentation

9.7.2.1 void update () [virtual]

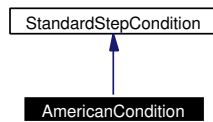
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.8 AmericanCondition Class Reference

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

Inheritance diagram for AmericanCondition:



9.8.1 Detailed Description

American exercise condition.

Todo

Unify the intrinsicValues/Payoff thing

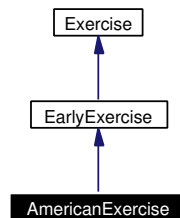
Public Member Functions

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

9.9 AmericanExercise Class Reference

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

Inheritance diagram for AmericanExercise:



9.9.1 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 the earliestDate and not earlier

Public Member Functions

- **AmericanExercise** (**Date** earliestDate, **Date** latestDate, bool payoffAtExpiry=false)

9.10 AmericanPayoffAtExpiry Class Reference

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

9.10.1 Detailed Description

Analytic formula for American exercise payoff at-expiry options

Todo

calculate greeks

Public Member Functions

- **AmericanPayoffAtExpiry** (double spot, double discount, double dividendDiscount, double variance, const **Handle**< **StrikedTypePayoff** > &payoff)
- double **value** () const

9.11 AmericanPayoffAtHit Class Reference

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

9.11.1 Detailed Description

Analytic formula for American exercise payoff at-hit options

Todo

calculate greeks

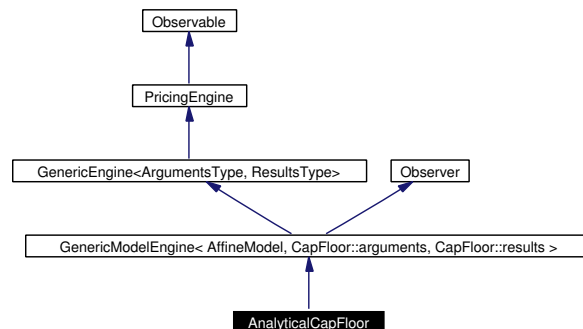
Public Member Functions

- **AmericanPayoffAtHit** (double spot, double discount, double dividendDiscount, double variance, const **Handle**< **StrikedTypePayoff** > &payoff)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **rho** (double maturity) const

9.12 AnalyticalCapFloor Class Reference

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

Inheritance diagram for AnalyticalCapFloor:



9.12.1 Detailed Description

Analytical pricer for cap/floor.

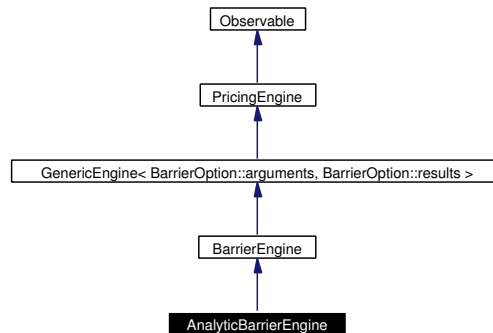
Public Member Functions

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

9.13 AnalyticBarrierEngine Class Reference

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

Inheritance diagram for AnalyticBarrierEngine:



9.13.1 Detailed Description

Pricing engine for barrier options using analytical formulae.

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

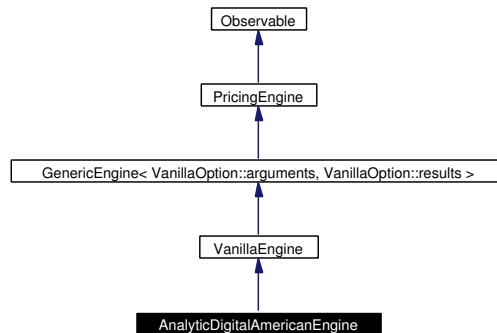
Public Member Functions

- void **calculate** () const

9.14 AnalyticDigitalAmericanEngine Class Reference

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

Inheritance diagram for AnalyticDigitalAmericanEngine:



9.14.1 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)

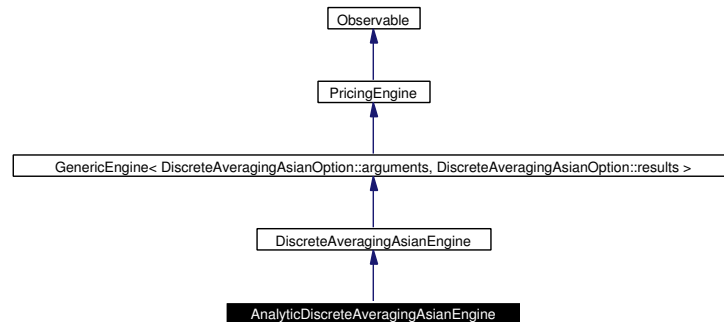
Public Member Functions

- void **calculate** () const

9.15 AnalyticDiscreteAveragingAsianEngine Class Reference

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

Inheritance diagram for AnalyticDiscreteAveragingAsianEngine:



9.15.1 Detailed Description

Pricing engine for European discrete geometric average Asian option.

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

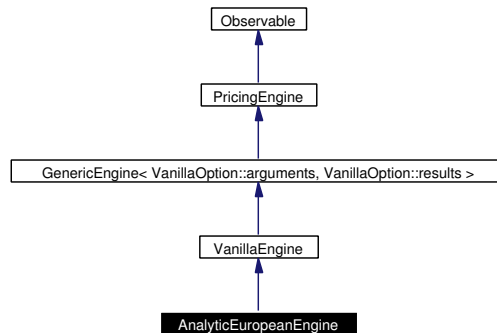
Public Member Functions

- `void calculate () const`

9.16 AnalyticEuropeanEngine Class Reference

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

Inheritance diagram for AnalyticEuropeanEngine:



9.16.1 Detailed Description

Pricing engine for European vanilla options using analytical formulae.

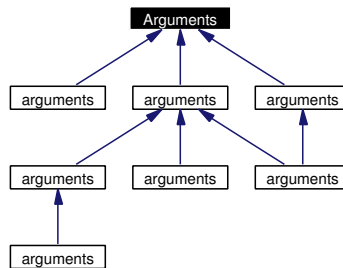
Public Member Functions

- void **calculate** () const

9.17 Arguments Class Reference

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

Inheritance diagram for Arguments:



9.17.1 Detailed Description

base class for generic argument groups

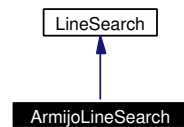
Public Member Functions

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

9.18 ArmijoLineSearch Class Reference

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

Inheritance diagram for ArmijoLineSearch:



9.18.1 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 stop when t verifies $f(x+t*d) - f(x) \leq -\alpha*t*f'(x+t*d)$ and $f(x+t/\beta*d) - f(x) > -\alpha*t*f'(x+t*d)/\beta$

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

Public Member Functions

- **ArmijoLineSearch** (double $\text{eps}=1\text{e-}8$, double $\alpha=0.5$, double $\beta=0.65$)
Default constructor.
- virtual **~ArmijoLineSearch** ()
Destructor.
- virtual double **operator()** (const **Problem** &P, double t_{ini})
Perform line search.

9.19 Array Class Reference

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

9.19.1 Detailed Description

1-D array used in linear algebra.

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

Public Types

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

Public Member Functions

- typedef **QL_REVERSE_ITERATOR** (iterator, double) **reverse_iterator**
- typedef **QL_REVERSE_ITERATOR** (const_iterator, double) **const_reverse_iterator**

Constructors, destructor, and assignment

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

- `const Array & operator/= (double)`

Element access

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

Inspectors

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

Iterator access

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

Utilities

- `void swap (Array &)`

Related Functions

(Note that these are not member functions.)

- `std::ostream & operator<< (std::ostream &, const Array &)`
- `double 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 &, double)`
- `const Disposable< Array > operator+ (double, const Array &)`
- `const Disposable< Array > operator- (const Array &, const Array &)`
- `const Disposable< Array > operator- (const Array &, double)`
- `const Disposable< Array > operator- (double, const Array &)`
- `const Disposable< Array > operator * (const Array &, const Array &)`
- `const Disposable< Array > operator * (const Array &, double)`
- `const Disposable< Array > operator * (double, const Array &)`
- `const Disposable< Array > operator/ (const Array &, const Array &)`
- `const Disposable< Array > operator/ (const Array &, double)`
- `const Disposable< Array > operator/ (double, 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 &)`

9.20 ArrayFormatter Class Reference

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

9.20.1 Detailed Description

Formats arrays for output.

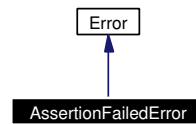
Static Public Member Functions

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

9.21 AssertionError Class Reference

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

Inheritance diagram for AssertionError:



9.21.1 Detailed Description

Specialized error.

Thrown upon a failed assertion.

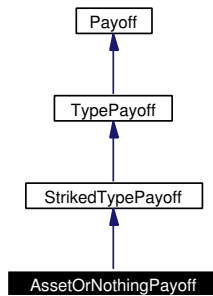
Public Member Functions

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

9.22 AssetOrNothingPayoff Class Reference

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

Inheritance diagram for AssetOrNothingPayoff:



9.22.1 Detailed Description

Binary asset-or-nothing payoff.

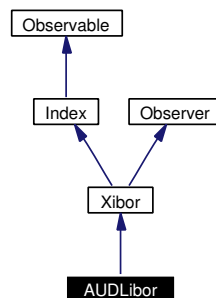
Public Member Functions

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

9.23 AUDLibor Class Reference

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

Inheritance diagram for AUDLibor:



9.23.1 Detailed Description

AUD Libor index, also known as SIBOR

Todo

check settlement days

Public Member Functions

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

9.24 Average Struct Reference

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

9.24.1 Detailed Description

placeholder for enumerated averaging types

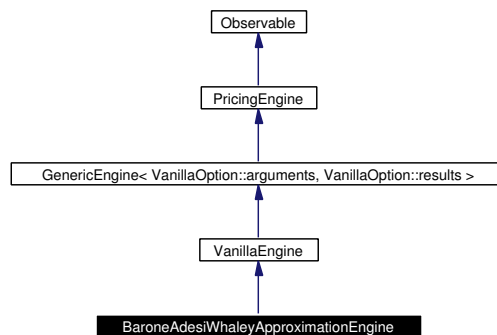
Public Types

- enum Type { Arithmetic, Geometric }

9.25 BaroneAdesiWhaleyApproximationEngine Class Reference

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

Inheritance diagram for BaroneAdesiWhaleyApproximationEngine:



9.25.1 Detailed Description

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

Public Member Functions

- void **calculate** () const

9.26 Barrier Struct Reference

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

9.26.1 Detailed Description

Placeholder for enumerated barrier types.

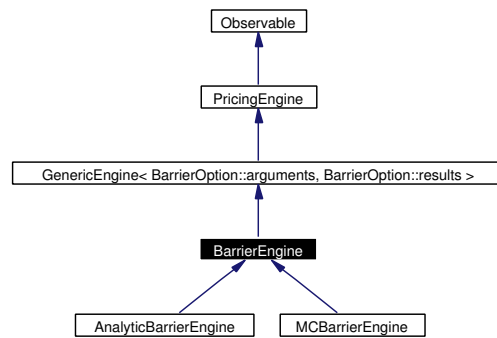
Public Types

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

9.27 BarrierEngine Class Reference

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

Inheritance diagram for BarrierEngine:



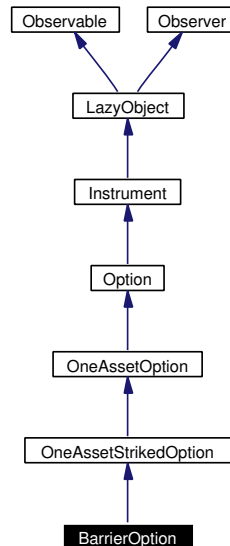
9.27.1 Detailed Description

Barrier engine base class

9.28 BarrierOption Class Reference

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

Inheritance diagram for BarrierOption:



9.28.1 Detailed Description

Barrier option on a single asset.

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

Public Member Functions

- **BarrierOption** (Barrier::Type barrierType, double barrier, double rebate, const **Handle**< BlackScholesStochasticProcess > &stochProc, const **Handle**< **StrikedTypePayoff** > &payoff, const **Handle**< **Exercise** > &exercise, const **Handle**< **PricingEngine** > &engine=**Handle**< **PricingEngine** >())
- void **setupArguments** (Arguments *) const

Protected Member Functions

- void **performCalculations** () const

Protected Attributes

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

9.28.2 Member Function Documentation

9.28.2.1 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** (p. [482](#)).

9.28.2.2 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** (p. [482](#)).

9.29 BarrierOption::arguments Class Reference

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

9.29.1 Detailed Description

Arguments for barrier option calculation

Public Member Functions

- void **validate** () const

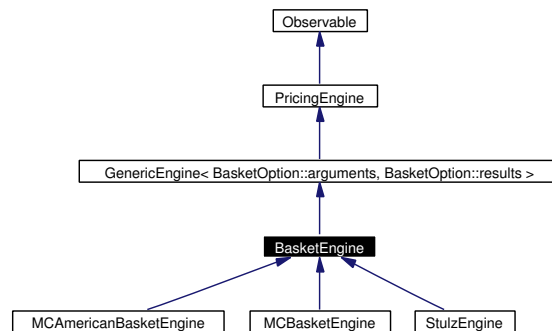
Public Attributes

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

9.30 BasketEngine Class Reference

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

Inheritance diagram for BasketEngine:



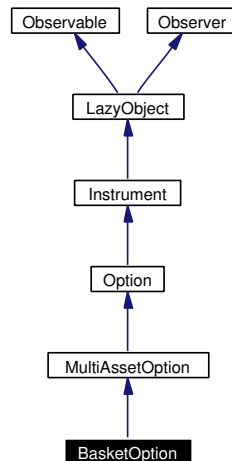
9.30.1 Detailed Description

Basket option engine base class

9.31 BasketOption Class Reference

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

Inheritance diagram for BasketOption:



9.31.1 Detailed Description

Basket option on a number of assets.

Public Types

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

Public Member Functions

- `BasketOption` (const `BasketType` basketType, const std::vector< `Handle`< `BlackScholesStochasticProcess` > > &stochProcs, const `Handle`< `PlainVanillaPayoff` > &payoff, const `Handle`< `Exercise` > &exercise, const `Matrix` &correlation, const `Handle`< `PricingEngine` > &engine=`Handle`< `PricingEngine` >())
- void `setupArguments` (`Arguments` *) const

Protected Member Functions

- void `performCalculations` () const

9.31.2 Member Function Documentation

9.31.2.1 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** (p. [452](#)).

9.31.2.2 void performCalculations () const [protected, virtual]

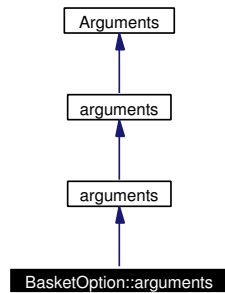
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **MultiAssetOption** (p. [452](#)).

9.32 BasketOption::arguments Class Reference

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

Inheritance diagram for BasketOption::arguments:



9.32.1 Detailed Description

Arguments for basket option calculation

Public Member Functions

- void **validate** () const

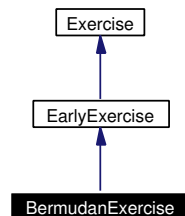
Public Attributes

- BasketType **basketType**

9.33 BermudanExercise Class Reference

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

Inheritance diagram for BermudanExercise:



9.33.1 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

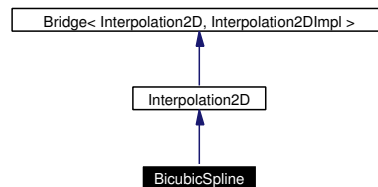
Public Member Functions

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

9.34 BicubicSpline Class Reference

#include <ql/Math/bicubicsplineinterpolation.hpp>

Inheritance diagram for BicubicSpline:



9.34.1 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)

9.34.2 Constructor & Destructor Documentation

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

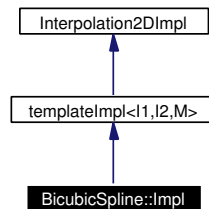
Precondition:

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

9.35 BicubicSpline::Impl Class Template Reference

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

Inheritance diagram for BicubicSpline::Impl:



9.35.1 Detailed Description

```
template<class I1, class I2, class M> class QuantLib::BicubicSpline::Impl< I1, I2, M >
```

bicubic spline implementation

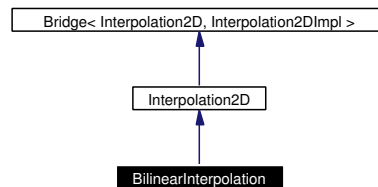
Public Member Functions

- **Impl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)
- double **value** (double x, double y) const

9.36 BilinearInterpolation Class Reference

#include <ql/Math/bilinearinterpolation.hpp>

Inheritance diagram for BilinearInterpolation:



9.36.1 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)

9.36.2 Constructor & Destructor Documentation

9.36.2.1 BilinearInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, const I2 & *yEnd*, const M & *zData*)

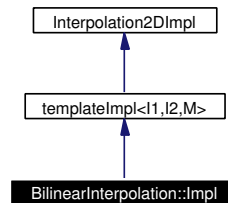
Precondition:

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

9.37 BilinearInterpolation::Impl Class Template Reference

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

Inheritance diagram for BilinearInterpolation::Impl:



9.37.1 Detailed Description

```
template<class I1, class I2, class M> class QuantLib::BilinearInterpolation::Impl< I1, I2, M >
```

bilinear interpolation implementation

Public Member Functions

- **Impl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)
- double **value** (double x, double y) const

9.38 BinomialDistribution Class Reference

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

9.38.1 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.

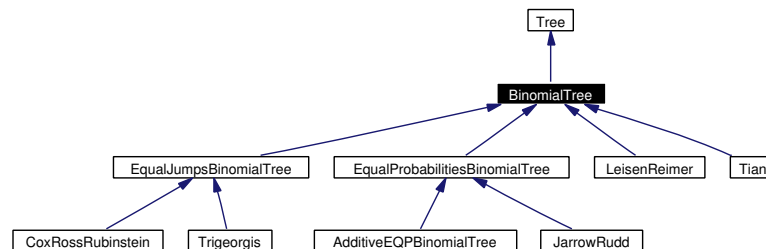
Public Member Functions

- **BinomialDistribution** (double p, unsigned long n)
- double **operator()** (unsigned long k) const

9.39 BinomialTree Class Reference

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

Inheritance diagram for BinomialTree:



9.39.1 Detailed Description

Binomial tree base class.

Public Member Functions

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

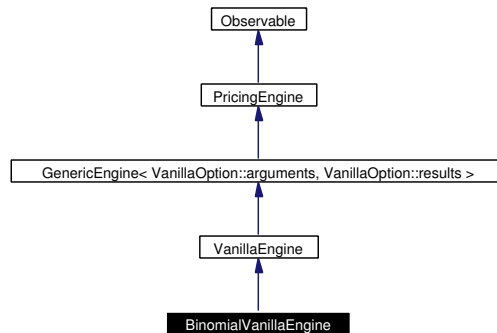
Protected Attributes

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

9.40 BinomialVanillaEngine Class Template Reference

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

Inheritance diagram for BinomialVanillaEngine:



9.40.1 Detailed Description

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

Pricing engine for vanilla options using binomial trees.

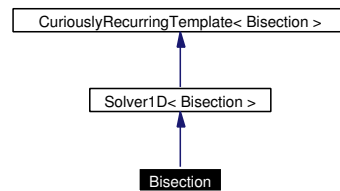
Public Member Functions

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

9.41 Bisection Class Reference

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

Inheritance diagram for Bisection:



9.41.1 Detailed Description

Bisection 1-D solver

Public Member Functions

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

9.42 BivariateCumulativeNormalDistribution Class Reference

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

9.42.1 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.

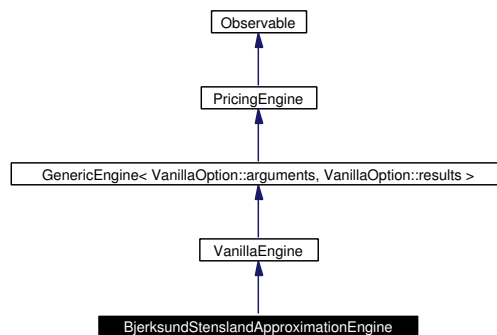
Public Member Functions

- **BivariateCumulativeNormalDistribution** (double rho)
- double **operator()** (double a, double b) const

9.43 BjerksundStenslandApproximationEngine Class Reference

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

Inheritance diagram for BjerksundStenslandApproximationEngine:



9.43.1 Detailed Description

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

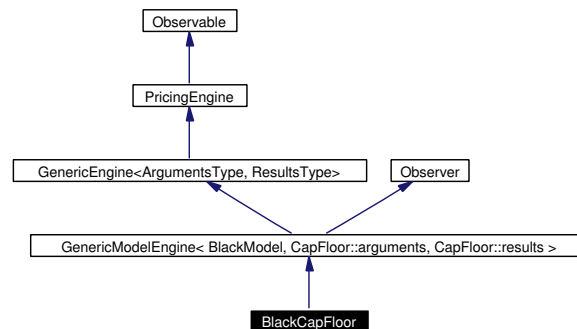
Public Member Functions

- void **calculate** () const

9.44 BlackCapFloor Class Reference

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

Inheritance diagram for BlackCapFloor:



9.44.1 Detailed Description

Cap/floor priced by means of the Black formula.

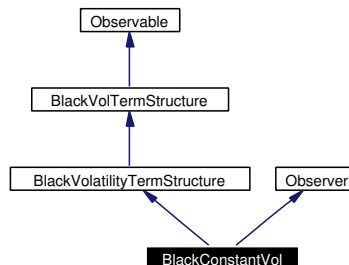
Public Member Functions

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

9.45 BlackConstantVol Class Reference

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

Inheritance diagram for BlackConstantVol:



9.45.1 Detailed Description

Constant Black volatility, no time-strike dependence.

This class implements the **BlackVolatilityTermStructure**(p. 169) interface for a constant Black volatility (no time/strike dependence).

Public Member Functions

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

BlackVolTermStructure interface

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

Observer interface

- void **update** ()

Visitability

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

Protected Member Functions

- virtual double **blackVolImpl** (Time t, double, bool extrapolate=false) const
implements the actual Black vol calculation in derived classes

9.45.2 Member Function Documentation

9.45.2.1 void update () [virtual]

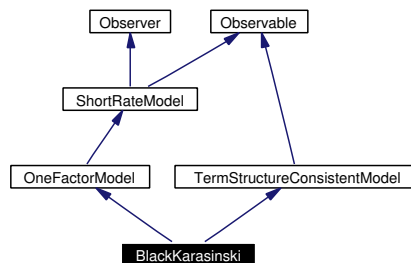
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.46 BlackKarasinski Class Reference

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

Inheritance diagram for BlackKarasinski:



9.46.1 Detailed Description

Standard Black-Karasinski model class.

This class implements the standard Black-Karasinski model defined by

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

where *alpha* and *sigma* are constants.

Public Member Functions

- **BlackKarasinski** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a=0.1, double sigma=0.1)
- **Handle**< **ShortRateDynamics** > **dynamics** () const
returns the short-rate dynamics
- **Handle**< **Lattice** > **tree** (const **TimeGrid** &grid) const
Return by default a trinomial recombining tree.

9.47 BlackKarasinski::Dynamics Class Reference

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

9.47.1 Detailed Description

Short-rate dynamics in the Black-Karasinski model.

The short-rate is here

$$r_t = e^{\varphi(t) + x_t}$$

where $\varphi(t)$ is the deterministic time-dependent parameter (which can not be determined analytically) used for term-structure fitting and x_t is the state variable following an Ornstein-Uhlenbeck process.

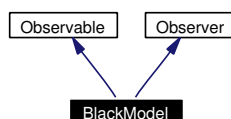
Public Member Functions

- **Dynamics** (const **Parameter** &fitting, double alpha, double sigma)
- double **variable** (Time t, Rate r) const
- double **shortRate** (Time t, double x) const

9.48 BlackModel Class Reference

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

Inheritance diagram for BlackModel:



9.48.1 Detailed Description

Black-model for vanilla interest-rate derivatives.

Public Member Functions

- **BlackModel** (const **RelinkableHandle**< **Quote** > &volatility, const **RelinkableHandle**< **TermStructure** > &termStructure)
- void **update** ()
- double **volatility** () const
- const **RelinkableHandle**< **TermStructure** > & **termStructure** () const

Static Public Member Functions

- double **formula** (double *f*, double *k*, double *v*, double *w*)
General Black formula.
- double **itmProbability** (double *f*, double *k*, double *v*, double *w*)
In-the-money cash probability.

9.48.2 Member Function Documentation

9.48.2.1 void update () [virtual]

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

Implements **Observer** (p. [475](#)).

9.48.2.2 double formula (double *f*, double *k*, double *v*, double *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

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

9.48.2.3 double itmProbability (double *f*, double *k*, double *v*, double *w*) [static]

In-the-money cash probability.

Returns

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

where

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

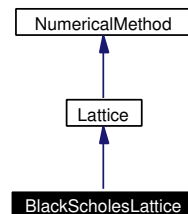
and

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

9.49 BlackScholesLattice Class Reference

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

Inheritance diagram for BlackScholesLattice:



9.49.1 Detailed Description

Simple binomial lattice approximating the Black-Scholes model.

Public Member Functions

- **BlackScholesLattice** (const **Handle**< **Tree** > &tree, Rate riskFreeRate, Time end, Size steps)
- Size **size** (Size i) const
- DiscountFactor **discount** (Size i, Size j) const
Discount factor at time t_i and node indexed by index.
- const **Handle**< **Tree** > & **tree** () const

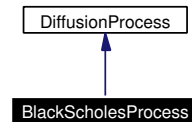
Protected Member Functions

- void **stepback** (Size i, const **Array** &values, **Array** &newValues) const
- Size **descendant** (Size i, Size index, Size branch) const
***Tree**(p. 607) properties.*
- double **probability** (Size i, Size index, Size branch) const

9.50 BlackScholesProcess Class Reference

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

Inheritance diagram for BlackScholesProcess:



9.50.1 Detailed Description

Black-Scholes diffusion process class.

This class describes the stochastic process governed by

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

Todo

revise extrapolation

Public Member Functions

- **BlackScholesProcess** (const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > ÷ndTS, const **RelinkableHandle**< **BlackVolTermStructure** > &blackVolTS, double s0)
- double **drift** (Time t, double x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- double **diffusion** (Time t, double x) const

9.50.2 Member Function Documentation

9.50.2.1 double diffusion (Time t, double x) const [virtual]

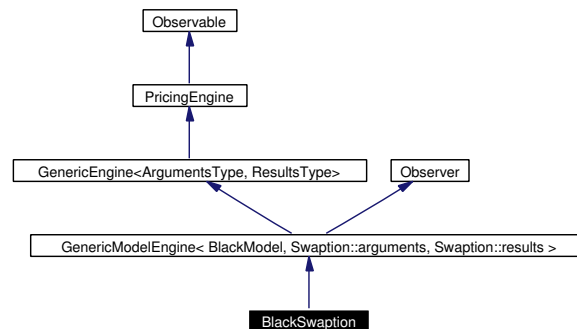
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

Implements **DiffusionProcess** (p. [246](#)).

9.51 BlackSwaption Class Reference

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

Inheritance diagram for BlackSwaption:



9.51.1 Detailed Description

Swaption priced by means of the Black formula

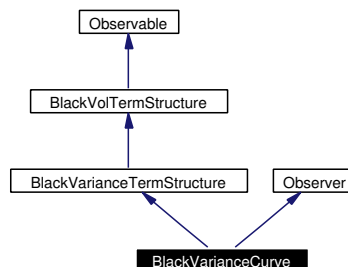
Public Member Functions

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

9.52 BlackVarianceCurve Class Reference

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

Inheritance diagram for BlackVarianceCurve:



9.52.1 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](#)(p. 400) interpolation is used as default; this can be changed by the `setInterpolation()` method.

For strike dependence, see [BlackVarianceSurface](#)(p. 166).

Todo

check time extrapolation

Public Member Functions

- **BlackVarianceCurve** (const **Date** &referenceDate, const std::vector< **Date** > &dates, const std::vector< double > &blackVolCurve, const **DayCounter** &dayCounter=**Actual365**())

BlackVolTermStructure interface

- **Date** **referenceDate** () const
returns the reference date for which t=0
- **DayCounter** **dayCounter** () const
returns the day counter
- **Date** **maxDate** () const
the latest date for which the term structure can return vols

Modifiers

- template<class Traits> void **setInterpolation** ()

Observer interface

- void **update** ()

Visitability

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

Protected Member Functions

- virtual double **blackVarianceImpl** (Time t, double, bool extrapolate=false) const
implements the actual Black variance calculation in derived classes

9.52.2 Member Function Documentation**9.52.2.1 void update () [virtual]**

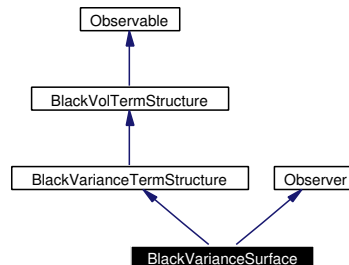
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.53 BlackVarianceSurface Class Reference

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

Inheritance diagram for BlackVarianceSurface:



9.53.1 Detailed Description

Black volatility surface modelled as variance surface.

This class calculates time/strike dependent Black volatilities using as input a matrix of Black volatilities observed in the market.

The calculation is performed interpolating on the variance surface. Bilinear interpolation is used as default; this can be changed by the `setInterpolation()` method.

Todo

check time extrapolation

Public Types

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

Public Member Functions

- BlackVarianceSurface** (const **Date** &referenceDate, const std::vector< **Date** > &dates, const std::vector< double > &strikes, const **Matrix** &blackVolMatrix, Extrapolation lowerExtrapolation=InterpolatorDefaultExtrapolation, Extrapolation upperExtrapolation=InterpolatorDefaultExtrapolation, const **DayCounter** &dayCounter=**Actual365**())

BlackVolTermStructure interface

- Date** **referenceDate** () const
returns the reference date for which t=0
- DayCounter** **dayCounter** () const
returns the day counter
- Date** **maxDate** () const
the latest date for which the term structure can return vols

Modifiers

- template<class Traits> void **setInterpolation** ()

Observer interface

- void **update** ()

Visitability

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

Protected Member Functions

- virtual double **blackVarianceImpl** (Time t, double strike, bool extrapolate=false) const
implements the actual Black variance calculation in derived classes

9.53.2 Member Function Documentation

9.53.2.1 void update () [virtual]

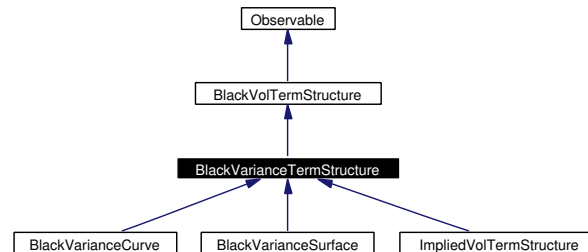
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.54 BlackVarianceTermStructure Class Reference

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

Inheritance diagram for BlackVarianceTermStructure:



9.54.1 Detailed Description

Black variance term structure.

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

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Visitability

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

Protected Member Functions

- double **blackVolImpl** (Time maturity, double strike, bool extrapolate=false) const

9.54.2 Member Function Documentation

9.54.2.1 double blackVolImpl (Time *maturity*, double *strike*, bool *extrapolate* = false) const [protected, virtual]

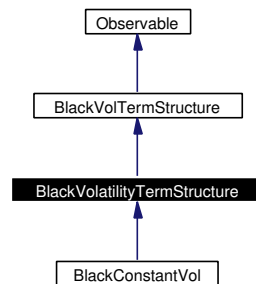
Returns the volatility for the given strike and date calculating it from the variance.

Implements **BlackVolTermStructure** (p. 171).

9.55 BlackVolatilityTermStructure Class Reference

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

Inheritance diagram for BlackVolatilityTermStructure:



9.55.1 Detailed Description

Black-volatility term structure.

This abstract class acts as an adapter to **BlackVolTermStructure**(p. 170) allowing the programmer to implement only the `blackVolImpl(Time, double, bool)` method in derived classes.

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Visitability

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

Protected Member Functions

- double **blackVarianceImpl** (Time maturity, double strike, bool extrapolate=false) const

9.55.2 Member Function Documentation

9.55.2.1 double **blackVarianceImpl** (Time *maturity*, double *strike*, bool *extrapolate* = false)
const [protected, virtual]

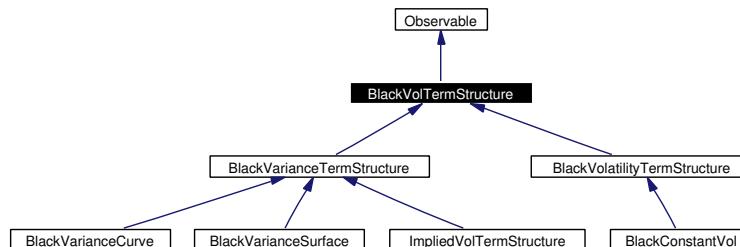
Returns the variance for the given strike and date calculating it from the volatility.

Implements **BlackVolTermStructure** (p. 171).

9.56 BlackVolTermStructure Class Reference

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

Inheritance diagram for BlackVolTermStructure:



9.56.1 Detailed Description

Black-volatility term structure.

This abstract class defines the interface of concrete Black-volatility term structures which will be derived from this one.

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

Black Volatility

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

future (a.k.a. forward) variance

- double **blackForwardVariance** (Time time1, Time time2, double strike, bool extrapolate=false) const
future (a.k.a. forward) variance

Dates

- virtual **Date referenceDate** () const=0
returns the reference date for which $t=0$
- virtual **DayCounter dayCounter** () const=0
returns the day counter
- virtual **Date maxDate** () const=0
the latest date for which the term structure can return vols
- Time **maxTime** () const
the latest time for which the term structure can return vols

Visitability

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

Protected Member Functions

- virtual double **blackVarianceImpl** (Time t, double strike, bool extrapolate=false) const=0
implements the actual Black variance calculation in derived classes
- virtual double **blackVolImpl** (Time t, double strike, bool extrapolate=false) const=0
implements the actual Black vol calculation in derived classes

9.57 BoundaryCondition Class Template Reference

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

9.57.1 Detailed Description

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

Abstract boundary condition class for finite difference problems.

Public Types

- typedef Operator **operatorType**
- typedef Operator::arrayType **arrayType**
- enum **Side** { None, Upper, Lower }

Public Member Functions

- virtual void **applyBeforeApplying** (operatorType &) const=0
- virtual void **applyAfterApplying** (arrayType &) const=0
- virtual void **applyBeforeSolving** (operatorType &, arrayType &rhs) const=0
- virtual void **applyAfterSolving** (arrayType &) const=0
- virtual void **setTime** (Time t)=0

9.57.2 Member Enumeration Documentation

9.57.2.1 enum Side

Todo

Generalize for n-dimensional conditions

9.57.3 Member Function Documentation

9.57.3.1 virtual void applyBeforeApplying (operatorType &) const [pure virtual]

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

9.57.3.2 virtual void applyAfterApplying (arrayType &) const [pure virtual]

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

9.57.3.3 virtual void applyBeforeSolving (operatorType &, arrayType & rhs) const [pure virtual]

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

9.57.3.4 virtual void applyAfterSolving (arrayType &) const [pure virtual]

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

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

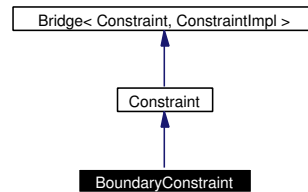
This method sets the current time for time-dependent boundary conditions.

Implemented in **NeumannBC** (p. [460](#)), and **DirichletBC** (p. [248](#)).

9.58 BoundaryConstraint Class Reference

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

Inheritance diagram for BoundaryConstraint:



9.58.1 Detailed Description

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

Public Member Functions

- **BoundaryConstraint** (double low, double high)

9.59 BoxMullerGaussianRng Class Template Reference

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

9.59.1 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< double > sample_type`

Public Member Functions

- `BoxMullerGaussianRng` (const RNG &uniformGenerator)
- `BoxMullerGaussianRng` (long seed=0)
- `sample_type next ()` const

returns next sample from the Gaussian distribution

9.59.2 Constructor & Destructor Documentation

9.59.2.1 BoxMullerGaussianRng (long seed = 0) [explicit]

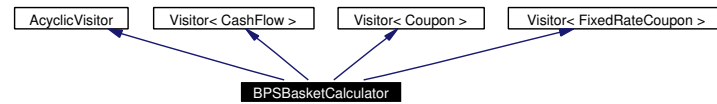
Deprecated

initialize with a random number generator instead.

9.60 BPSBasketCalculator Class Reference

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

Inheritance diagram for BPSBasketCalculator:



9.60.1 Detailed Description

Bug

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

Public Member Functions

- **BPSBasketCalculator** (const **RelinkableHandle**< **TermStructure** > &ts, int basis)
- const **TimeBasket** & **result** () const

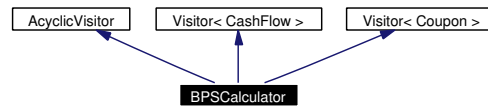
Visitor interface

- double **sensfactor** (const **Date** &date) const
- virtual void **visit** (**Coupon** &)
- virtual void **visit** (**FixedRateCoupon** &)
- virtual void **visit** (**CashFlow** &)

9.61 BPSCalculator Class Reference

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

Inheritance diagram for BPSCalculator:



9.61.1 Detailed Description

basis point sensitivity (BPS) calculator

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

Public Member Functions

- **BPSCalculator** (const **RelinkableHandle**< **TermStructure** > &ts)
- double **result** () const

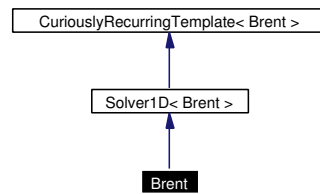
Visitor interface

- virtual void **visit** (**Coupon** &)
- virtual void **visit** (**CashFlow** &)

9.62 Brent Class Reference

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

Inheritance diagram for Brent:



9.62.1 Detailed Description

Brent 1-D solver

Public Member Functions

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

9.63 Bridge Class Template Reference

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

9.63.1 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 Handle< Impl > &impl=Handle< Impl >())

Protected Attributes

- Handle< Impl > impl_

9.64 BrownianBridge Class Template Reference

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

9.64.1 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

Public Types

- typedef **Sample**< std::vector< double > > **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< double > &sigma, const **TimeGrid** &timeGrid, const GSG &generator)
general Wiener process paths
- **BrownianBridge** (const **Handle**< **BlackVolTermStructure** > &blackVol, const **TimeGrid** &timeGrid, const GSG &generator)
- **BrownianBridge** (const **Handle**< **DiffusionProcess** > &diffProcess, const **TimeGrid** &timeGrid, const GSG &generator)

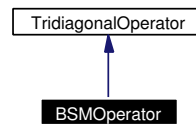
inspectors

- const **sample_type** & **next** () const
- const **sample_type** & **last** () const
- Size **size** () const
- const **TimeGrid** & **timeGrid** () const

9.65 BSMOperator Class Reference

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

Inheritance diagram for BSMOperator:



9.65.1 Detailed Description

Black-Scholes-Merton differential operator.

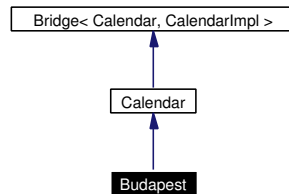
Public Member Functions

- **BSMOperator** (Size size, double dx, double r, double q, double sigma)

9.66 Budapest Class Reference

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

Inheritance diagram for Budapest:



9.66.1 Detailed Description

Budapest calendar

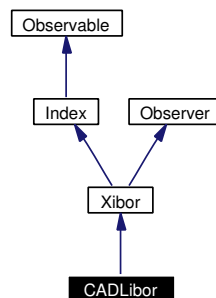
Holidays:

- 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

9.67 CADLibor Class Reference

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

Inheritance diagram for CADLibor:



9.67.1 Detailed Description

CAD Libor index, also known as CDOR

Todo

check settlement days

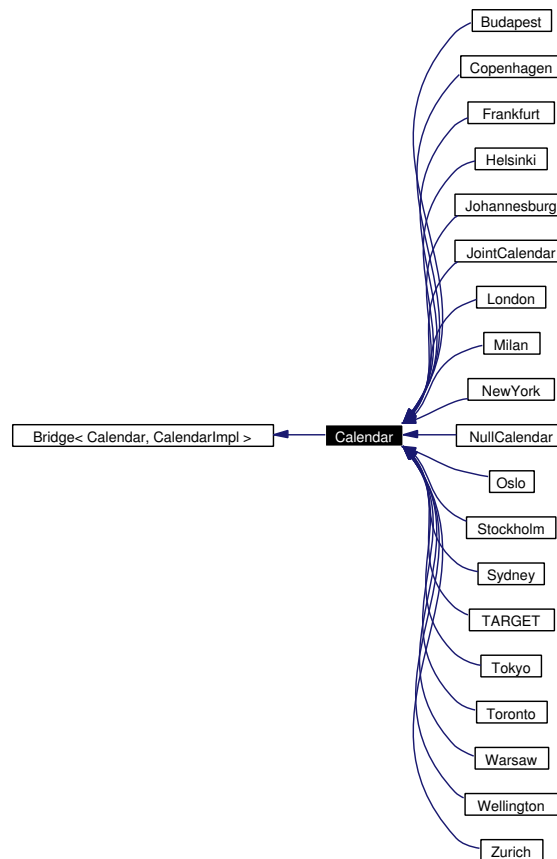
Public Member Functions

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

9.68 Calendar Class Reference

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

Inheritance diagram for Calendar:



9.68.1 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**([p. 179](#)) pattern is used to provide the base behavior of the calendar, namely, to determine whether a date is a business day.

Public Member Functions

- `Calendar ()`

Calendar interface

- `std::string name () const`
Returns the name of the calendar.

- **bool isBusinessDay** (const **Date** &d) const
- **bool isEndOfMonth** (const **Date** &d) const
- **bool isHoliday** (const **Date** &d) const
- **Date roll** (const **Date** &, RollingConvention convention=Following, const **Date** &origin=**Date**()) const
- **Date advance** (const **Date** &, int n, TimeUnit unit, RollingConvention convention=Following) const
- **Date advance** (const **Date** &date, const **Period** &period, RollingConvention convention) const

Protected Member Functions

- **Calendar** (const **Handle**< **CalendarImpl** > &impl)

Related Functions

(Note that these are not member functions.)

- **bool operator==** (const **Calendar** &, const **Calendar** &)
- **bool operator!=** (const **Calendar** &, const **Calendar** &)

9.68.2 Constructor & Destructor Documentation

9.68.2.1 Calendar ()

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

9.68.2.2 Calendar (const **Handle**< **CalendarImpl** > &impl) [protected]

This protected constructor will only be invoked by derived classes which define a given **Calendar**(p. 184) implementation

9.68.3 Member Function Documentation

9.68.3.1 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.

9.68.3.2 bool isBusinessDay (const **Date** &d) const

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

9.68.3.3 bool isEndOfMonth (const Date & *d*) const

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

9.68.3.4 bool isHoliday (const Date & *d*) const

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

9.68.3.5 Date roll (const Date &, RollingConvention *convention* = Following, const Date & *origin* = Date()) const

Returns the next business day on the given market with respect to the given date and convention.

9.68.3.6 Date advance (const Date &, int *n*, TimeUnit *unit*, RollingConvention *convention* = Following) const

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

Note:

The input date is not modified.

9.68.3.7 Date advance (const Date & *date*, const Period & *period*, RollingConvention *convention*) const

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

Note:

The input date is not modified.

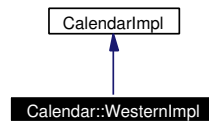
9.68.4 Friends And Related Function Documentation**9.68.4.1 bool operator== (const Calendar &, const Calendar &) [related]**

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

9.69 Calendar::WesternImpl Class Reference

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

Inheritance diagram for Calendar::WesternImpl:



9.69.1 Detailed Description

partial calendar implementation

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

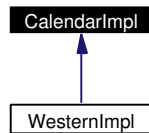
Static Protected Member Functions

- Day **easterMonday** (Year y)
expressed relative to first day of year

9.70 CalendarImpl Class Reference

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

Inheritance diagram for CalendarImpl:



9.70.1 Detailed Description

abstract base class for calendar implementations

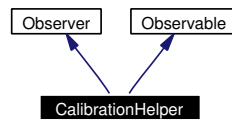
Public Member Functions

- virtual `std::string name () const=0`
- virtual `bool isBusinessDay (const Date &) const=0`

9.71 CalibrationHelper Class Reference

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

Inheritance diagram for CalibrationHelper:



9.71.1 Detailed Description

liquid market instrument used during calibration

Public Member Functions

- **CalibrationHelper** (const **RelinkableHandle**< **Quote** > &volatility, const **RelinkableHandle**< **TermStructure** > &termStructure)
- void **update** ()
- double **marketValue** ()
returns the actual price of the instrument (from volatility)
- virtual double **modelValue** ()=0
returns the price of the instrument according to the model
- virtual double **calibrationError** ()
returns the error resulting from the model valuation
- virtual void **addTimesTo** (std::list< **Time** > ×) const=0
- double **impliedVolatility** (double targetValue, double accuracy, Size maxEvaluations, double minVol, double maxVol) const
Black volatility implied by the model.
- virtual double **blackPrice** (double volatility) const=0
Black price given a volatility.
- void **setPricingEngine** (const **Handle**< **PricingEngine** > &engine)

Protected Attributes

- double **marketValue_**
- **RelinkableHandle**< **Quote** > **volatility_**
- **RelinkableHandle**< **TermStructure** > **termStructure_**
- **Handle**< **BlackModel** > **blackModel_**
- **Handle**< **PricingEngine** > **engine_**

9.71.2 Member Function Documentation

9.71.2.1 `void update ()` [virtual]

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

Implements **Observer** (p. [475](#)).

9.72 CalibrationSet Class Reference

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

9.72.1 Detailed Description

Set of calibration instruments.

For the moment, this is just here to facilitate the assignment of a pricing engine to a set of calibration helpers

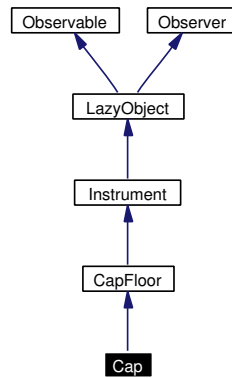
Public Member Functions

- void **setPricingEngine** (const **Handle**< **PricingEngine** > &engine)

9.73 Cap Class Reference

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

Inheritance diagram for Cap:



9.73.1 Detailed Description

Concrete cap class.

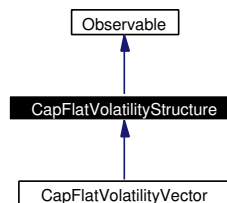
Public Member Functions

- **Cap** (const std::vector< **Handle**< **CashFlow** > > &floatingLeg, const std::vector< **Rate** > &exerciseRates, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)

9.74 CapFlatVolatilityStructure Class Reference

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

Inheritance diagram for CapFlatVolatilityStructure:



9.74.1 Detailed Description

Cap/floor flat volatility structure.

This class is purely abstract and defines the interface of concrete structures which will be derived from this one.

Public Member Functions

- virtual **Date** **todayDate** () const=0
returns today's date
- virtual **Date** **settlementDate** () const=0
returns the settlement date
- virtual **DayCounter** **dayCounter** () const=0
returns the day counter used for internal date/time conversions
- double **volatility** (const **Date** &end, Rate strike) const
returns the volatility for a given end date and strike rate
- double **volatility** (const **Period** &length, Rate strike) const
returns the volatility for a given cap/floor length and strike rate
- double **volatility** (Time t, Rate strike) const
returns the volatility for a given end time and strike rate

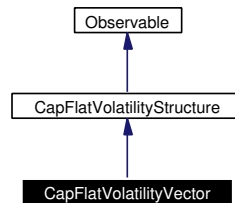
Protected Member Functions

- virtual double **volatilityImpl** (Time length, Rate strike) const=0
implements the actual volatility calculation in derived classes

9.75 CapFlatVolatilityVector Class Reference

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

Inheritance diagram for CapFlatVolatilityVector:



9.75.1 Detailed Description

Cap/floor at-the-money flat volatility vector.

This class provides the at-the-money volatility for a given cap by interpolating a volatility vector whose elements are the market volatilities of a set of caps/floors with given length.

Todo

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

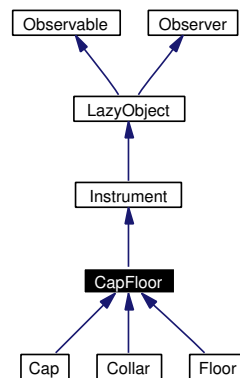
Public Member Functions

- **CapFlatVolatilityVector** (const **Date** &todayDate, const **Calendar** &calendar, int settlementDays, const std::vector< **Period** > &lengths, const std::vector< double > &volatilities, const **DayCounter** &dayCounter=**Thirty360**())
- **Date todayDate** () const
returns today's date
- **Date settlementDate** () const
returns the settlement date
- **DayCounter dayCounter** () const
returns the day counter used for internal date/time conversions

9.76 CapFloor Class Reference

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

Inheritance diagram for CapFloor:



9.76.1 Detailed Description

Base class for cap-like instruments.

Public Types

- enum **Type** { **Cap**, **Floor**, **Collar** }

Public Member Functions

- **CapFloor** (Type type, const std::vector< **Handle**< **CashFlow** > > &floatingLeg, const std::vector< Rate > &capRates, const std::vector< Rate > &floorRates, const **Relinkable-Handle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)
- void **setupArguments** (**Arguments** *) const
- double **impliedVolatility** (double price, double accuracy=1.0e-4, Size maxEvaluations=100, double minVol=QL_MIN_VOLATILITY, double maxVol=QL_MAX_VOLATILITY) const
implied term volatility

Instrument interface

- bool **isExpired** () const
returns whether the instrument is still tradable.

Inspectors

- Type **type** () const
- const std::vector< **Handle**< **CashFlow** > > & **leg** () const
- const std::vector< Rate > & **capRates** () const
- const std::vector< Rate > & **floorRates** () const

9.76.2 Member Function Documentation

9.76.2.1 `void setupArguments (Arguments *) const` [virtual]

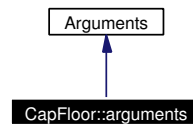
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **Instrument** (p. [368](#)).

9.77 CapFloor::arguments Class Reference

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

Inheritance diagram for CapFloor::arguments:



9.77.1 Detailed Description

Arguments for cap/floor calculation

Public Member Functions

- void **validate** () const

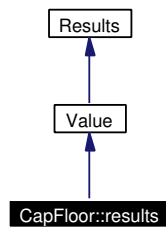
Public Attributes

- CapFloor::Type **type**
- std::vector< Time > **startTimes**
- std::vector< Time > **fixingTimes**
- std::vector< Time > **endTimes**
- std::vector< Time > **accrualTimes**
- std::vector< Rate > **capRates**
- std::vector< Rate > **floorRates**
- std::vector< Rate > **forwards**
- std::vector< double > **nominals**

9.78 CapFloor::results Class Reference

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

Inheritance diagram for CapFloor::results:



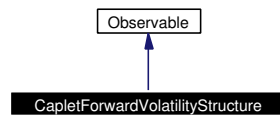
9.78.1 Detailed Description

Results from cap/floor calculation

9.79 CapletForwardVolatilityStructure Class Reference

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

Inheritance diagram for CapletForwardVolatilityStructure:



9.79.1 Detailed Description

Caplet/floorlet forward volatility structure.

This class is purely abstract and defines the interface of concrete structures which will be derived from this one.

Public Member Functions

- virtual **Date** **todaysDate** () const=0
returns today's date
- virtual **Date** **settlementDate** () const=0
returns the settlement date
- virtual **DayCounter** **dayCounter** () const=0
returns the day counter used for internal date/time conversions
- double **volatility** (const **Date** &start, Rate strike) const
returns the volatility for a given start date and strike rate
- double **volatility** (Time t, Rate strike) const
returns the volatility for a given start time and strike rate

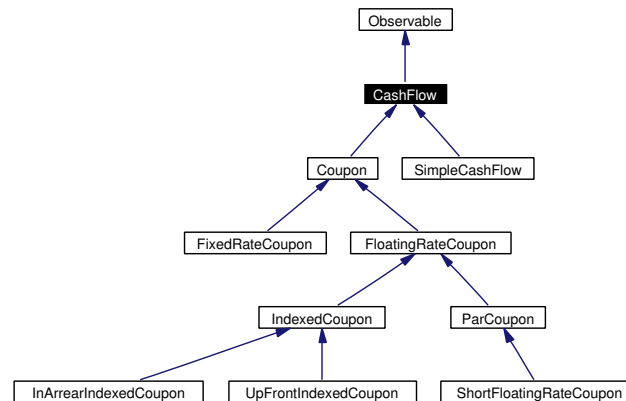
Protected Member Functions

- virtual double **volatilityImpl** (Time length, Rate strike) const=0
implements the actual volatility calculation in derived classes

9.80 CashFlow Class Reference

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

Inheritance diagram for CashFlow:



9.80.1 Detailed Description

Base class for cash flows.

This class is purely virtual and acts as a base class for the actual cash flow implementations.

Public Member Functions

CashFlow interface

- virtual double **amount** () const=0
returns the amount of the cash flow
- virtual **Date** **date** () const=0
returns the date at which the cash flow is settled

Visitability

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

9.80.2 Member Function Documentation

9.80.2.1 virtual double amount () const [pure virtual]

returns the amount of the cash flow

Note:

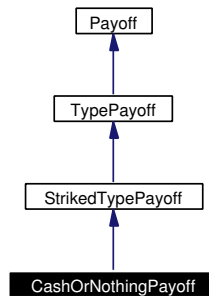
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implemented in **FixedRateCoupon** (p. 304), **IndexedCoupon** (p. 365), **ParCoupon** (p. 500), **ShortFloatingRateCoupon** (p. 549), and **SimpleCashFlow** (p. 553).

9.81 CashOrNothingPayoff Class Reference

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

Inheritance diagram for CashOrNothingPayoff:



9.81.1 Detailed Description

Binary cash-or-nothing payoff.

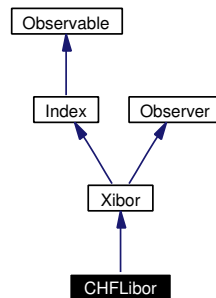
Public Member Functions

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

9.82 CHFLibor Class Reference

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

Inheritance diagram for CHFLibor:



9.82.1 Detailed Description

CHF Libor index, also known as ZIBOR

Todo

check settlement days and day-count

Public Member Functions

- **CHFLibor** (int n, TimeUnit units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**Actual360**())

9.83 CLGaussianRng Class Template Reference

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

9.83.1 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< double > sample_type`

Public Member Functions

- `CLGaussianRng (const RNG &uniformGenerator)`
- `CLGaussianRng (long seed=0)`
- `sample_type next () const`

returns next sample from the Gaussian distribution

9.83.2 Constructor & Destructor Documentation

9.83.2.1 CLGaussianRng (long seed = 0) [explicit]

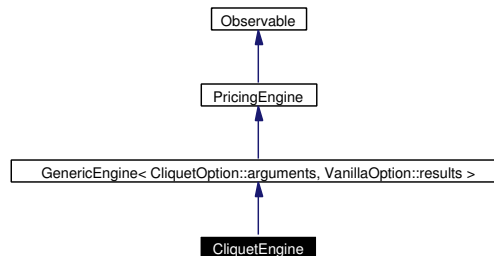
Deprecated

initialize with a random number generator instead.

9.84 CliquetEngine Class Reference

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

Inheritance diagram for CliquetEngine:



9.84.1 Detailed Description

Cliquet engine base class.

9.85 CliquetOption::arguments Class Reference

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

9.85.1 Detailed Description

Arguments for cliquet option calculation

Public Member Functions

- void **validate** () const

Public Attributes

- double **moneyiness**
- double **accruedCoupon**
- double **lastFixing**
- double **localCap**
- double **localFloor**
- double **globalCap**
- double **globalFloor**
- std::vector< **Date** > **resetDates**

9.86 CliquetOptionPricer Class Reference

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

9.86.1 Detailed Description

cliquet (Ratchet) option

A cliquet option, also known as Ratchet option, is a series of forward-starting (a.k.a. deferred strike) options where the strike for each forward start option is set equal to a fixed percentage of the spot price at the beginning of each period.

In the particular case in which only two dates are given the cliquet option is the same as a forward-starting option starting at the first date and expiring at the second date.

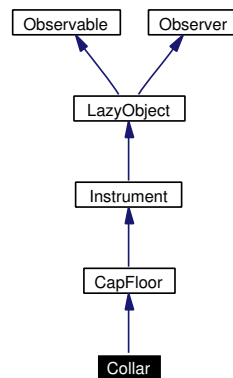
Public Member Functions

- **CliquetOptionPricer** (Option::Type type, double underlying, double moneyness, const std::vector< Spread > ÷ndYield, const std::vector< Rate > &riskFreeRate, const std::vector< Time > ×, const std::vector< double > &volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const

9.87 Collar Class Reference

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

Inheritance diagram for Collar:



9.87.1 Detailed Description

Concrete collar class.

Public Member Functions

- **Collar** (const std::vector< **Handle**< **CashFlow** > > &floatingLeg, const std::vector< Rate > &capRates, const std::vector< Rate > &floorRates, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)

9.88 combining_iterator Class Template Reference

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

9.88.1 Detailed Description

template<class Iterator, class Function> class QuantLib::combining_iterator< Iterator, Function >

Iterator mapping a function to a set of underlying sequences.

This iterator advances a set of underlying iterators and returns the values obtained by applying a function to the sets of values such iterators point to.

This class was implemented based on Christopher Baus and Thomas Becker, *Custom Iterators for the STL*, included in the proceedings of the First Workshop on C++ Template Programming, Erfurt, Germany, 2000 (<http://www.oonumerics.org/tmpw00/>)

Public Types

- typedef Function::result_type **value_type**
- typedef const Function::result_type * **pointer**
- typedef const Function::result_type & **reference**

Public Member Functions

- template<class IteratorCollectionIterator> **combining_iterator** (IteratorCollectionIterator it1, IteratorCollectionIterator it2, Function f)

Dereferencing

- reference **operator*** () const
- pointer **operator** → () const

Random access

- value_type **operator**[] (difference_type n) const

Increment and decrement

- **combining_iterator** & **operator++** ()
- **combining_iterator** **operator++** (int)
- **combining_iterator** & **operator--** ()
- **combining_iterator** **operator--** (int)
- **combining_iterator** & **operator+=** (difference_type n)
- **combining_iterator** & **operator-=** (difference_type n)
- **combining_iterator** **operator+** (difference_type n) const
- **combining_iterator** **operator-** (difference_type n) const

Difference

- difference_type **operator-** (const **combining_iterator**< Iterator, Function > &rhs) const

Comparisons

- `bool operator== (const combining_iterator< Iterator, Function > &rhs) const`
- `bool operator!= (const combining_iterator< Iterator, Function > &rhs) const`

Public Attributes

- `typedef< Iterator >::difference_type difference_type`

Related Functions

(Note that these are not member functions.)

- `combining_iterator< typename 1< It >::value_type, Function > make_combining_iterator`
(It it1, It it2, Function f)
helper function to create combining iterators

9.89 Composite Class Template Reference

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

9.89.1 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< **Handle**< T > >::iterator **iterator**
- typedef std::list< **Handle**< T > >::const_iterator **const_iterator**

Protected Member Functions

- void **add** (const **Handle**< T > &c)

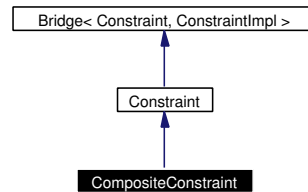
Protected Attributes

- std::list< **Handle**< T > > **components_**

9.90 CompositeConstraint Class Reference

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

Inheritance diagram for CompositeConstraint:



9.90.1 Detailed Description

Constraint enforcing both given sub-constraints

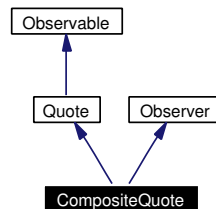
Public Member Functions

- **CompositeConstraint** (const **Constraint** &c1, const **Constraint** &c2)

9.91 CompositeQuote Class Template Reference

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

Inheritance diagram for CompositeQuote:



9.91.1 Detailed Description

```
template<class BinaryFunction> class QuantLib::CompositeQuote< BinaryFunction >
```

market element whose value depends on two other market element

Public Member Functions

- **CompositeQuote** (const **RelinkableHandle**< **Quote** > &element1, const **RelinkableHandle**< **Quote** > &element2, const BinaryFunction &f)

Quote interface

- double **value** () const
returns the current value

Observer interface

- void **update** ()

9.91.2 Member Function Documentation

9.91.2.1 void update () [virtual]

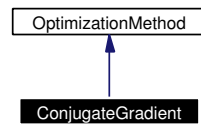
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 475).

9.92 ConjugateGradient Class Reference

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

Inheritance diagram for ConjugateGradient:



9.92.1 Detailed Description

Multi-dimensional Conjugate Gradient class.

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

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

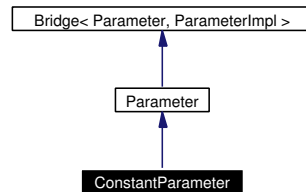
Public Member Functions

- **ConjugateGradient** ()
default constructor
- **ConjugateGradient** (const **Handle**< **LineSearch** > &lineSearch)
- virtual **~ConjugateGradient** ()
destructor
- virtual void **minimize** (const **Problem** &P) const
minimize the optimization problem P

9.93 ConstantParameter Class Reference

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

Inheritance diagram for ConstantParameter:



9.93.1 Detailed Description

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

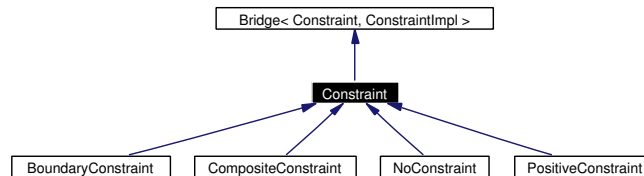
Public Member Functions

- **ConstantParameter** (const **Constraint** &constraint)
- **ConstantParameter** (double value, const **Constraint** &constraint)

9.94 Constraint Class Reference

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

Inheritance diagram for Constraint:



9.94.1 Detailed Description

Base constraint class.

Public Member Functions

- `bool test (const Array &p) const`
- `double update (Array &p, const Array &direction, double beta)`
- `Constraint (const Handle< ConstraintImpl > &impl=Handle< ConstraintImpl >())`

9.95 ConstraintImpl Class Reference

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

9.95.1 Detailed Description

Base class for constraint implementations.

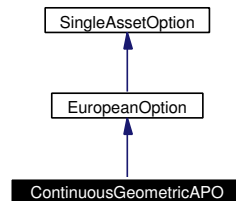
Public Member Functions

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

9.96 ContinuousGeometricAPO Class Reference

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

Inheritance diagram for ContinuousGeometricAPO:



9.96.1 Detailed Description

Continuous geometric average-price option (European exercise).

This class implements a continuous geometric average price asian option with european exercise. The formula is from "Option Pricing Formulas", E. G. Haug (1997) pag 96-97

Todo

add **Average**(p. [132](#)) Strike version and make it backward starting

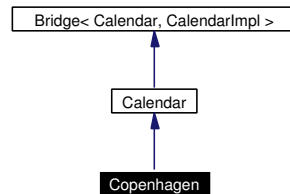
Public Member Functions

- **ContinuousGeometricAPO** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility)
- double **vega** () const
- double **rho** () const
- **Handle< SingleAssetOption > clone** () const

9.97 Copenhagen Class Reference

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

Inheritance diagram for Copenhagen:



9.97.1 Detailed Description

Copenhagen calendar

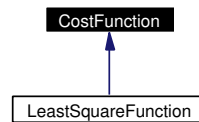
Holidays:

- 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

9.98 CostFunction Class Reference

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

Inheritance diagram for CostFunction:



9.98.1 Detailed Description

Cost function abstract class for optimization problem.

Public Member Functions

- virtual double **value** (const **Array** &x) const=0
method to overload to compute the cost 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 double **valueAndGradient** (**Array** &grad, const **Array** &x) const
method to overload to compute grad_f, the first derivative of
- virtual double **finiteDifferenceEpsilon** () const
Default epsilon for finite difference method .:

9.99 coupling_iterator Class Template Reference

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

9.99.1 Detailed Description

```
template<class Iterator1, class Iterator2, class Function> class QuantLib::coupling_iterator<
Iterator1, Iterator2, Function >
```

Iterator mapping a function to a pair of underlying sequences.

This iterator advances two underlying iterators and returns the values obtained by applying a function to the two values such iterators point to.

Public Types

- typedef Function::result_type **value_type**
- typedef const Function::result_type * **pointer**
- typedef const Function::result_type & **reference**

Public Member Functions

- **coupling_iterator** (Iterator1 it1, Iterator2 it2, Function f)

Dereferencing

- reference **operator *** () const
- pointer **operator →** () const

Random access

- **value_type operator[]** (difference_type n) const

Increment and decrement

- **coupling_iterator & operator++** ()
- **coupling_iterator operator++** (int)
- **coupling_iterator & operator--** ()
- **coupling_iterator operator--** (int)
- **coupling_iterator & operator+=** (difference_type n)
- **coupling_iterator & operator-=** (difference_type n)
- **coupling_iterator operator+** (difference_type n) const
- **coupling_iterator operator-** (difference_type n) const

Difference

- difference_type **operator-** (const **coupling_iterator** &rhs) const

Comparisons

- bool **operator==** (const **coupling_iterator** &rhs) const
- bool **operator!=** (const **coupling_iterator** &rhs) const

Public Attributes

- `typedef< Iterator1 >::difference_type` **difference_type**

Related Functions

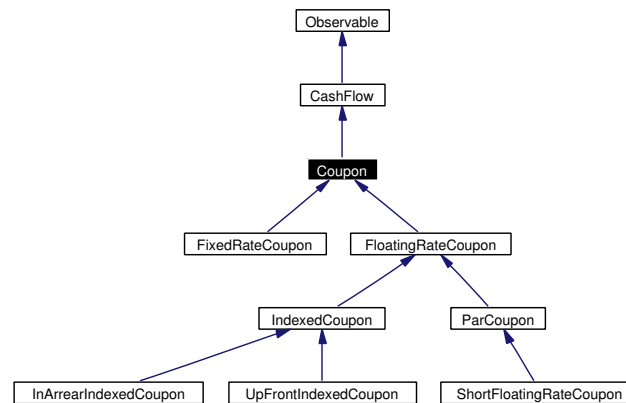
(Note that these are not member functions.)

- `coupling_iterator< It1, It2, Function > make_coupling_iterator` (It1 it1, It2 it2, Function f)
helper function to create combining iterators

9.100 Coupon Class Reference

#include <ql/CashFlows/coupon.hpp>

Inheritance diagram for Coupon:



9.100.1 Detailed Description

coupon accruing over a fixed period

This class implements part of the **CashFlow**(p. 200) interface but it is still abstract and provides derived classes with methods for accrual period calculations.

Public Member Functions

- **Coupon** (double nominal, const **Date** &paymentDate, const **Date** &accrualStartDate, const **Date** &accrualEndDate, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())

Partial CashFlow interface

- **Date** **date** () const
returns the date at which the cash flow is settled

Inspectors

- double **nominal** () const
- const **Date** & **accrualStartDate** () const
start of the accrual period
- const **Date** & **accrualEndDate** () const
end of the accrual period
- Time **accrualPeriod** () const
accrual period as fraction of year

- **int** **accrualDays** () const
accrual period in days
- **virtual DayCounter** **dayCounter** () const=0
day counter for accrual calculation
- **virtual double** **accruedAmount** (const **Date** &) const=0
accrued amount at the given date

Visitability

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

Protected Attributes

- **double** **nominal_**
- **Date** **paymentDate_**
- **Date** **accrualStartDate_**
- **Date** **accrualEndDate_**
- **Date** **refPeriodStart_**
- **Date** **refPeriodEnd_**

9.100.2 Constructor & Destructor Documentation

- 9.100.2.1 **Coupon** (double *nominal*, const **Date** & *paymentDate*, const **Date** & *accrualStartDate*, const **Date** & *accrualEndDate*, const **Date** & *refPeriodStart* = **Date**(), const **Date** & *refPeriodEnd* = **Date**())

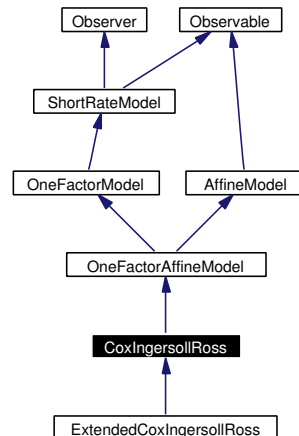
Warning:

the coupon does not roll the payment date which must already be a business day.

9.101 CoxIngersollRoss Class Reference

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

Inheritance diagram for CoxIngersollRoss:



9.101.1 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, double theta=0.1, double k=0.1, double sigma=0.1)
- virtual double **discountBondOption** (Option::Type type, double strike, Time maturity, Time bondMaturity) const
- virtual **Handle**< ShortRateDynamics > **dynamics** () const
returns the short-rate dynamics
- virtual **Handle**< Lattice > **tree** (const TimeGrid &grid) const
Return by default a trinomial recombining tree.

Protected Member Functions

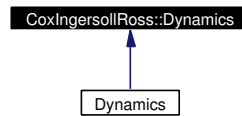
- double **A** (Time t, Time T) const
- double **B** (Time t, Time T) const
- double **theta** () const
- double **k** () const

- double **sigma** () const
- double **x0** () const

9.102 CoxIngersollRoss::Dynamics Class Reference

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

Inheritance diagram for CoxIngersollRoss::Dynamics:



9.102.1 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

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

.

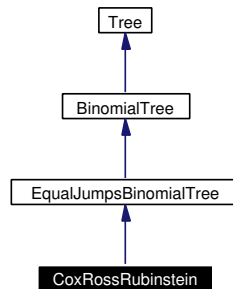
Public Member Functions

- **Dynamics** (double theta, double k, double sigma, double x0)
- virtual double **variable** (Time t, Rate r) const
- virtual double **shortRate** (Time t, double y) const

9.103 CoxRossRubinstein Class Reference

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

Inheritance diagram for CoxRossRubinstein:



9.103.1 Detailed Description

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

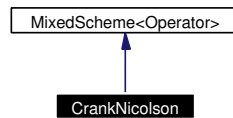
Public Member Functions

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

9.104 CrankNicolson Class Template Reference

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

Inheritance diagram for CrankNicolson:



9.104.1 Detailed Description

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

Crank-Nicolson scheme for finite difference methods.

See sect. **The finite differences framework**(p. 37) for details on the method.

In this implementation, the passed operator must be derived from either `TimeConstantOperator` or `TimeDependentOperator`. Also, it must implement at least the following interface:

```
typedef ... arrayType;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
arrayType applyTo(const arrayType&);
arrayType solveFor(const arrayType&);
static Operator identity(Size size);

// operator algebra
Operator operator*(double, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Warning:

The differential operator must be linear for this evolver to work.

Friends

- class `FiniteDifferenceModel< CrankNicolson< Operator > >`

9.105 Cubic Class Reference

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

9.105.1 Detailed Description

Cubic-spline interpolation traits.

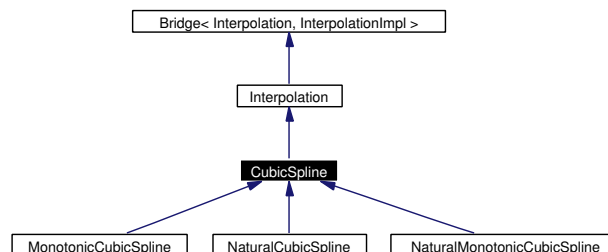
Static Public Member Functions

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

9.106 CubicSpline Class Reference

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

Inheritance diagram for CubicSpline:



9.106.1 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**(p. 229) and Quintic Hermite Interpolation" Mathematics Of Computation, v. 52, n. 186, April 1989, pp. 471-494.

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, double leftConditionValue, **CubicSpline::BoundaryCondition** rightCondition, double rightConditionValue, bool monotonicityConstraint)
- const std::vector< double > & **aCoefficients** () const
- const std::vector< double > & **bCoefficients** () const
- const std::vector< double > & **cCoefficients** () const

9.106.2 Member Enumeration Documentation

9.106.2.1 enum BoundaryCondition

Enumeration values:

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

9.106.3 Constructor & Destructor Documentation

9.106.3.1 **CubicSpline** (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, CubicSpline::BoundaryCondition *leftCondition*, double *leftConditionValue*, CubicSpline::BoundaryCondition *rightCondition*, double *rightConditionValue*, bool *monotonicityConstraint*)

Precondition:

the *x* values must be sorted.

9.107 CumulativeBinomialDistribution Class Reference

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

9.107.1 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** (double p , unsigned long n)
- double **operator()** (unsigned long k) const

9.108 CumulativeNormalDistribution Class Reference

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

9.108.1 Detailed Description

Cumulative normal distribution function.

Given x it provides an approximation to the integral of the gaussian normal distribution: formula here ...

For this implementation see M. Abramowitz and I. Stegun, Handbook of Mathematical Functions, Dover Publications, New York (1972)

Public Member Functions

- **CumulativeNormalDistribution** (double average=0.0, double sigma=1.0)
- double **operator()** (double x) const
- double **derivative** (double x) const

9.109 CumulativePoissonDistribution Class Reference

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

9.109.1 Detailed Description

Cumulative Poisson distribution function.

Given x it provides an approximation to the integral of the Poisson distribution: formula here ...

For this implementation see "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery, chapter 6

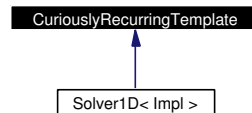
Public Member Functions

- **CumulativePoissonDistribution** (double μ)
- double **operator()** (unsigned long k) const

9.110 CuriouslyRecurringTemplate Class Template Reference

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

Inheritance diagram for CuriouslyRecurringTemplate:



9.110.1 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

9.111 CurrencyFormatter Class Reference

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

9.111.1 Detailed Description

Formats currencies for output.

Static Public Member Functions

- `std::string toString (Currency c)`

9.112 Date Class Reference

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

9.112.1 Detailed Description

Concrete date class.

This class provides methods to inspect dates as well as methods and operators which implement a limited date algebra (increasing and decreasing dates, and calculating their difference).

Public Member Functions

constructors

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

inspectors

- Weekday **weekday ()** const
- Day **dayOfMonth ()** const
- bool **isEndOfMonth ()** const
- Day **lastDayOfMonth ()** const
- Day **dayOfYear ()** const
One-based (Jan 1st = 1).
- Month **month ()** const
- Year **year ()** const
- long **serialNumber ()** const

date algebra

- **Date & operator+= (int days)**
increments date in place
- **Date & operator-= (int days)**
decrement date in place
- **Date & operator++ ()**
1-day pre-increment
- **Date operator++ (int)**
1-day post-increment

- **Date & operator-** ()
1-day pre-decrement
- **Date operator-** (int)
1-day post-decrement
- **Date operator+** (int days) const
returns a new incremented date
- **Date operator-** (int days) const
returns a new decremented date

other methods to increment/decrement dates

- **Date plusDays** (int days) const
- **Date plusWeeks** (int weeks) const
- **Date plusMonths** (int months) const
- **Date plusYears** (int years) const
- **Date plus** (int units, TimeUnit) const
- **Date plus** (const Period &) const

Static Public Member Functions

static methods

- bool **isLeap** (Year y)
- **Date minDate** ()
earliest allowed date
- **Date maxDate** ()
latest allowed date
- **Date todaysDate** ()
today's date.

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator<<** (std::ostream &, const Date &)
- long **operator-** (const Date &, const Date &)
Difference in days between dates.
- bool **operator==** (const Date &, const Date &)
- bool **operator!=** (const Date &, const Date &)
- bool **operator<** (const Date &, const Date &)
- bool **operator<=** (const Date &, const Date &)
- bool **operator>** (const Date &, const Date &)
- bool **operator>=** (const Date &, const Date &)

9.113 DateFormatter Class Reference

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

9.113.1 Detailed Description

Formats dates for output.

Formatting can be in short (mm/dd/yyyy) or long (Month ddth, yyyy) form.

Public Types

- enum Format { Long, Short, ISO }

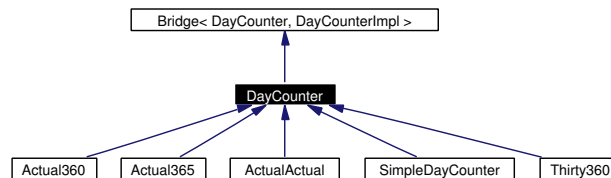
Static Public Member Functions

- std::string toString (const Date &d, Format f=Long)

9.114 DayCounter Class Reference

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

Inheritance diagram for DayCounter:



9.114.1 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**(p. 179) pattern is used to provide the base behavior of the day counter.

Public Member Functions

- **DayCounter** ()

DayCounter interface

- **std::string name** () const
Returns the name of the day counter.
- **int dayCount** (const **Date** &, const **Date** &) const
Returns the number of days between two dates.
- **Time yearFraction** (const **Date** &, const **Date** &, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**()) const
Returns the period between two dates as a fraction of year.

Protected Member Functions

- **DayCounter** (const **Handle**< **DayCounterImpl** > &impl)

Related Functions

(Note that these are not member functions.)

- **bool operator==** (const **DayCounter** &, const **DayCounter** &)
- **bool operator!=** (const **DayCounter** &, const **DayCounter** &)

9.114.2 Constructor & Destructor Documentation

9.114.2.1 DayCounter ()

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

9.114.2.2 DayCounter (const Handle< DayCounterImpl > & impl) [protected]

This protected constructor will only be invoked by derived classes which define a given **DayCounter**(p. 240) implementation

9.114.3 Member Function Documentation

9.114.3.1 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.

9.114.4 Friends And Related Function Documentation

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

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

9.115 DayCounterImpl Class Reference

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

9.115.1 Detailed Description

abstract base class for day counter implementations

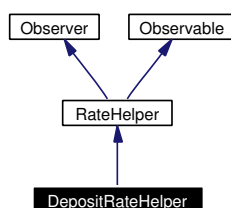
Public Member Functions

- virtual std::string **name** () const=0
- virtual int **dayCount** (const **Date** &, const **Date** &) const=0
- virtual Time **yearFraction** (const **Date** &, const **Date** &, const **Date** &refPeriodStart, const **Date** &refPeriodEnd) const=0

9.116 DepositRateHelper Class Reference

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

Inheritance diagram for DepositRateHelper:



9.116.1 Detailed Description

Deposit rate.

Warning:

This class assumes that the reference date does not change between calls of `setTermStructure()`(p. 243).

Public Member Functions

- **DepositRateHelper** (const **RelinkableHandle**< **Quote** > &rate, int n, TimeUnit units, int settlementDays, const **Calendar** &calendar, RollingConvention convention, const **DayCounter** &dayCounter)
- **DepositRateHelper** (double rate, int n, TimeUnit units, int settlementDays, const **Calendar** &calendar, RollingConvention convention, const **DayCounter** &dayCounter)
- double **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- void **setTermStructure** (**TermStructure** *)
sets the term structure to be used for pricing
- **Date maturity** () const
maturity date

9.116.2 Member Function Documentation

9.116.2.1 void setTermStructure (TermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:

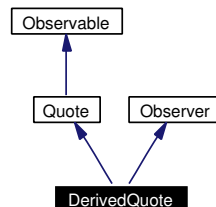
Being a pointer and not a **Handle**(p. 339), the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from **RateHelper** (p. [537](#)).

9.117 DerivedQuote Class Template Reference

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

Inheritance diagram for DerivedQuote:



9.117.1 Detailed Description

```
template<class UnaryFunction> class QuantLib::DerivedQuote< UnaryFunction >
```

market element whose value depends on another market element

Public Member Functions

- **DerivedQuote** (const RelinkableHandle< Quote > &element, const UnaryFunction &f)

Market element interface

- double **value** () const
returns the current value

Observer interface

- void **update** ()

9.117.2 Member Function Documentation

9.117.2.1 void update () [virtual]

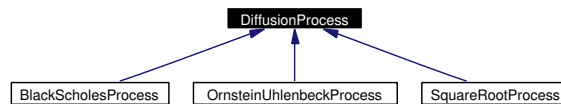
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 475).

9.118 DiffusionProcess Class Reference

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

Inheritance diagram for DiffusionProcess:



9.118.1 Detailed Description

Diffusion process class.

This class describes a stochastic process governed by

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

Public Member Functions

- **DiffusionProcess** (double x0)
- double **x0** () const
- virtual double **drift** (Time t, double x) const=0
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- virtual double **diffusion** (Time t, double x) const=0
- virtual double **expectation** (Time t0, double x0, Time dt) const
returns the expectation of the process after a time interval
- virtual double **variance** (Time t0, double x0, Time dt) const
returns the variance of the process after a time interval

9.118.2 Member Function Documentation

9.118.2.1 virtual double diffusion (Time t, double x) const [pure virtual]

returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

Implemented in **BlackScholesProcess** (p. 162), **OrnsteinUhlenbeckProcess** (p. 493), and **SquareRootProcess** (p. 567).

9.118.2.2 virtual double expectation (Time t0, double x0, Time dt) const [virtual]

returns the expectation of the process after a time interval

returns $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$. By default, it returns the Euler approximation defined by $x_0 + \mu(t_0, x_0)\Delta t$.

Reimplemented in **OrnsteinUhlenbeckProcess** (p. 493).

9.118.2.3 virtual double variance (Time $t0$, double $x0$, Time dt) const [virtual]

returns the variance of the process after a time interval

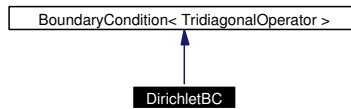
returns $Var(x_{t_0+\Delta t}|x_{t_0} = x_0)$. By default, it returns the Euler approximation defined by $\sigma(t_0, x_0)^2 \Delta t$.

Reimplemented in **OrnsteinUhlenbeckProcess** (p. [494](#)).

9.119 DirichletBC Class Reference

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

Inheritance diagram for DirichletBC:



9.119.1 Detailed Description

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

Todo

generalize to time-dependent conditions.

Public Member Functions

- **DirichletBC** (double value, Side side)
- void **applyBeforeApplying** (TridiagonalOperator &) const
- void **applyAfterApplying** (Array &) const
- void **applyBeforeSolving** (TridiagonalOperator &, Array &rhs) const
- void **applyAfterSolving** (Array &) const
- void **setTime** (Time t)

9.119.2 Member Function Documentation

9.119.2.1 void setTime (Time t) [virtual]

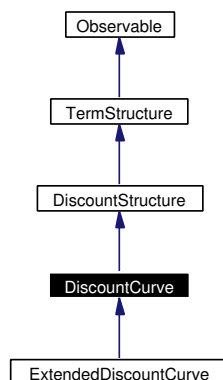
This method sets the current time for time-dependent boundary conditions.

Implements **BoundaryCondition< TridiagonalOperator >** (p. 173).

9.120 DiscountCurve Class Reference

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

Inheritance diagram for DiscountCurve:



9.120.1 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

- **DiscountCurve** (const **Date** &today'sDate, const std::vector< **Date** > &dates, const std::vector< DiscountFactor > &dfs, const **DayCounter** &dayCounter=**Actual365**())
- **Date** today'sDate () const
today's date
- **Date** referenceDate () const
the reference date, i.e., the date at which discount = 1
- **DayCounter** dayCounter () const
the day counter used for date/time conversion
- **Date** maxDate () const
the latest date for which the curve can return rates
- Time maxTime () const
the latest time for which the curve can return rates
- const std::vector< Time > & times () const
- const std::vector< **Date** > & dates () const
- const std::vector< DiscountFactor > & discounts () const

Protected Member Functions

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

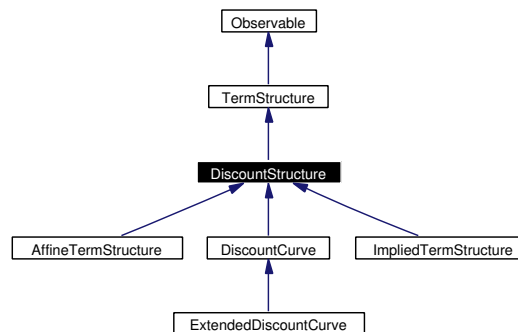
Protected Attributes

- Date **todaysDate_**
- Date **referenceDate_**
- DayCounter **dayCounter_**
- std::vector< Date > **dates_**
- std::vector< DiscountFactor > **discounts_**
- std::vector< Time > **times_**
- Interpolation **interpolation_**

9.121 DiscountStructure Class Reference

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

Inheritance diagram for DiscountStructure:



9.121.1 Detailed Description

Discount factor term structure.

This abstract class acts as an adapter to [TermStructure](#)(p. 594) allowing the programmer to implement only the `discountImpl(const Date&, bool)` method in derived classes.

Rates are assumed to be annual continuous compounding.

Protected Member Functions

- Rate **zeroYieldImpl** (Time, bool *extrapolate*=false) const
- Rate **forwardImpl** (Time, bool *extrapolate*=false) const
- Rate **compoundForwardImpl** (Time, int, bool *extrapolate*=false) const

9.121.2 Member Function Documentation

9.121.2.1 Rate **zeroYieldImpl** (Time, bool *extrapolate* = false) const [protected, virtual]

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

Implements [TermStructure](#) (p. 595).

9.121.2.2 Rate **forwardImpl** (Time, bool *extrapolate* = false) const [protected, virtual]

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

Implements [TermStructure](#) (p. 595).

9.121.2.3 Rate compoundForwardImpl (Time, int, bool *extrapolate* = false) const
[protected, virtual]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Implements **TermStructure** (p. [595](#)).

Reimplemented in **ExtendedDiscountCurve** (p. [289](#)).

9.122 DiscrepancyStatistics Class Reference

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

Inheritance diagram for DiscrepancyStatistics:



9.122.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

It inherit from `SequenceStatistics<Statistics>`(p. 546) and adds L^2 discrepancy calculation

Public Member Functions

- **DiscrepancyStatistics** (Size dimension)
- `template<class Sequence> void add (const Sequence &sample, double weight=1.0)`
- `template<class Iterator> void add (Iterator begin, Iterator end, double weight=1.0)`
- `void reset (Size dimension=0)`

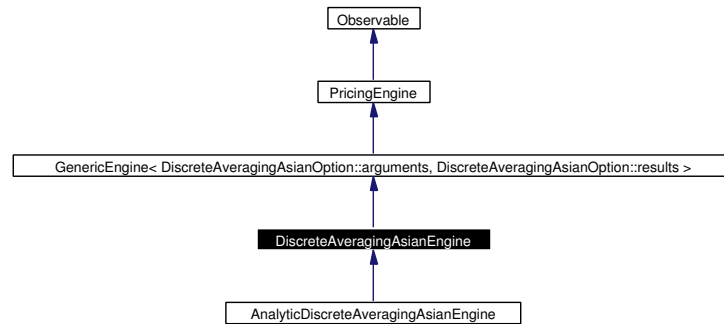
1-dimensional inspectors

- `double discrepancy () const`

9.123 DiscreteAveragingAsianEngine Class Reference

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

Inheritance diagram for DiscreteAveragingAsianEngine:



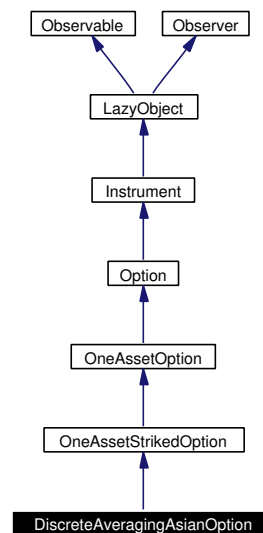
9.123.1 Detailed Description

Discrete averaging asian engine base class.

9.124 DiscreteAveragingAsianOption Class Reference

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

Inheritance diagram for DiscreteAveragingAsianOption:



9.124.1 Detailed Description

Asian option.

Public Member Functions

- **DiscreteAveragingAsianOption** (Average::Type averageType, double runningProduct, Size pastFixings, std::vector< Date > fixingDates, const Handle< BlackScholesStochasticProcess > &stochProc, const Handle< StrikedTypePayoff > &payoff, const Handle< Exercise > &exercise, const Handle< PricingEngine > &engine=Handle< PricingEngine >())
- void **setupArguments** (Arguments *) const

Protected Member Functions

- void **performCalculations** () const

Protected Attributes

- Average::Type **averageType_**
- double **runningProduct_**
- Size **pastFixings_**
- std::vector< Date > **fixingDates_**

9.124.2 Member Function Documentation

9.124.2.1 `void setupArguments (Arguments *) const` [virtual]

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

Reimplemented from **OneAssetStrikedOption** (p. [482](#)).

9.124.2.2 `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** (p. [482](#)).

9.125 DiscreteAveragingAsianOption::arguments Class Reference

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

9.125.1 Detailed Description

extra arguments for single asset asian option calculation

Public Member Functions

- void **validate** () const

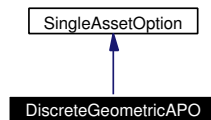
Public Attributes

- Average::Type **averageType**
- double **runningProduct**
- Size **pastFixings**
- std::vector< Date > **fixingDates**

9.126 DiscreteGeometricAPO Class Reference

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

Inheritance diagram for DiscreteGeometricAPO:



9.126.1 Detailed Description

Discrete geometric average-price Asian option (European style).

This class implements a discrete geometric average price asian option, with european exercise. The formula is from "Asian Option", E. Levy (1997) in "Exotic Options: The State of the Art", edited by L. Clewlow, C. Strickland, pag65-97

Todo

add analytical greeks

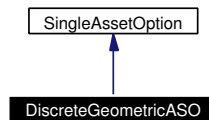
Public Member Functions

- **DiscreteGeometricAPO** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, const std::vector< Time > ×, double volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- **Handle< SingleAssetOption > clone** () const

9.127 DiscreteGeometricASO Class Reference

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

Inheritance diagram for DiscreteGeometricASO:



9.127.1 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

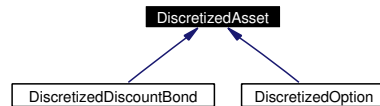
Public Member Functions

- **DiscreteGeometricASO** (Option::Type type, double underlying, Spread dividendYield, Rate riskFreeRate, const std::vector< Time > ×, double volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- **Handle< SingleAssetOption > clone** () const

9.128 DiscretizedAsset Class Reference

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

Inheritance diagram for DiscretizedAsset:



9.128.1 Detailed Description

Discretized asset class used by numerical methods.

Public Member Functions

- **DiscretizedAsset** (const **Handle**< **NumericalMethod** > &method)
- virtual void **reset** (Size size)=0
- Time **time** () const
- Time & **time** ()
- const **Array** & **values** () const
- **Array** & **values** ()
- const **Handle**< **NumericalMethod** > & **method** () const
- virtual void **preAdjustValues** ()
- virtual void **postAdjustValues** ()
- void **adjustValues** ()
- virtual void **addTimesTo** (std::list< Time > ×) const

Protected Member Functions

- bool **isOnTime** (Time t) const

Protected Attributes

- Time **time_**
- Array **values_**

9.128.2 Member Function Documentation

9.128.2.1 virtual void preAdjustValues () [virtual]

This method will be invoked after rollback and before any other asset (i.e., an option on this one) has any chance to look at the values. For instance, payments happening at times already spanned by the rollback will be added here.

9.128.2.2 virtual void postAdjustValues () [virtual]

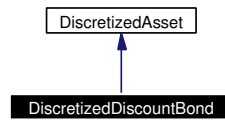
This method will be invoked after rollback and after any other asset had their chance to look at the values. For instance, payments happening at the present time (and therefore not included in an option to be exercised at this time) will be added here.

Reimplemented in **DiscretizedOption** (p. [263](#)).

9.129 DiscretizedDiscountBond Class Reference

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

Inheritance diagram for DiscretizedDiscountBond:



9.129.1 Detailed Description

Useful discretized discount bond asset.

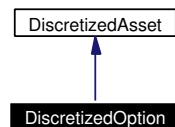
Public Member Functions

- **DiscretizedDiscountBond** (const **Handle**< **NumericalMethod** > &method)
- void **reset** (Size size)

9.130 DiscretizedOption Class Reference

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

Inheritance diagram for DiscretizedOption:



9.130.1 Detailed Description

Discretized option on another asset.

Precondition:

The underlying asset must be initialized

Public Member Functions

- **DiscretizedOption** (const **Handle**< **DiscretizedAsset** > &underlying, Exercise::Type exerciseType, const std::vector< Time > &exerciseTimes)
- void **reset** (Size size)
- void **postAdjustValues** ()
- void **addTimesTo** (std::list< Time > ×) const

Protected Member Functions

- void **applyExerciseCondition** ()

Protected Attributes

- **Handle**< **DiscretizedAsset** > **underlying_**
- Exercise::Type **exerciseType_**
- std::vector< Time > **exerciseTimes_**

9.130.2 Member Function Documentation

9.130.2.1 void postAdjustValues () [virtual]

This method will be invoked after rollback and after any other asset had their chance to look at the values. For instance, payments happening at the present time (and therefore not included in an option to be exercised at this time) will be added here.

Reimplemented from **DiscretizedAsset** (p. 261).

9.131 Disposable Class Template Reference

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

9.131.1 Detailed Description

template<class T> class QuantLib::Disposable< T >

generic disposable object with move semantics

This class can be used for returning a value by copy. It relies on the returned object exposing a `swap(T&)` method through which the copy constructor and assignment operator are implemented, thus resulting in actual move semantics. Typical use of this class is along the following lines:

```
Disposable<Foo> bar(int i) {  
    Foo f(i*2);  
    return f;  
}
```

Warning:

In order to avoid copies in code such as shown above, the conversion from T to **Disposable**(p. 264)<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;  
}
```

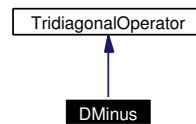
Public Member Functions

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

9.132 DMinus Class Reference

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

Inheritance diagram for DMinus:



9.132.1 Detailed Description

D_- matricial representation

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

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

Public Member Functions

- **DMinus** (Size gridPoints, double h)

9.133 DoubleFormatter Class Reference

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

9.133.1 Detailed Description

Formats doubles for output.

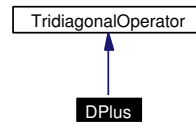
Static Public Member Functions

- `std::string toString` (double x, int precision=6, int digits=0)
- `std::string toExponential` (double x, int precision=6, int digits=0)

9.134 DPlus Class Reference

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

Inheritance diagram for DPlus:



9.134.1 Detailed Description

D_+ matricial representation

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

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

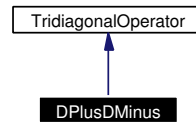
Public Member Functions

- **DPlus** (Size gridPoints, double h)

9.135 DPlusDMinus Class Reference

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

Inheritance diagram for DPlusDMinus:



9.135.1 Detailed Description

D_+D_- matricial representation

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

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

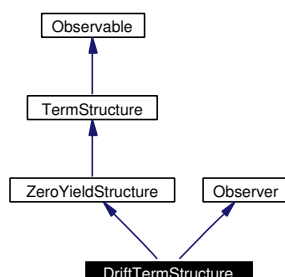
Public Member Functions

- **DPlusDMinus** (Size gridPoints, double h)

9.136 DriftTermStructure Class Reference

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

Inheritance diagram for DriftTermStructure:



9.136.1 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 **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > ÷ndTS, const **RelinkableHandle**< **BlackVolTermStructure** > &blackVolTS)

TermStructure interface

- **DayCounter dayCounter** () const
the day counter used for date/time conversion
- **Date todaysDate** () const
today's date
- **Date referenceDate** () const
the reference date, i.e., the date at which discount = 1
- **Date maxDate** () const
the latest date for which the curve can return rates

Observer interface

- **void update** ()

Protected Member Functions

- Rate **zeroYieldImpl** (Time, bool extrapolate=false) const
returns the discount factor as seen from the evaluation date

9.136.2 Member Function Documentation

9.136.2.1 void update () [virtual]

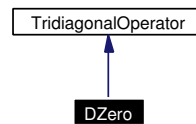
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.137 DZero Class Reference

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

Inheritance diagram for DZero:



9.137.1 Detailed Description

D_0 matricial representation

The differential operator D_0 discretizes the first derivative with the second-order formula

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

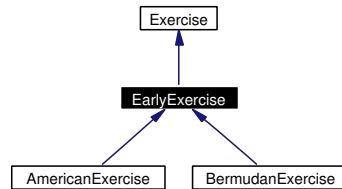
Public Member Functions

- **DZero** (Size gridPoints, double h)

9.138 EarlyExercise Class Reference

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

Inheritance diagram for EarlyExercise:



9.138.1 Detailed Description

Early-exercise base class.

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

Todo

derive a plain American **Exercise**(p. 283) class (no earliestDate, no payoffAtExpiry)

Public Member Functions

- **EarlyExercise** (bool payoffAtExpiry=false)
- bool **payoffAtExpiry** () const

9.139 EndCriteria Class Reference

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

9.139.1 Detailed Description

Criteria to end optimization process.

- stationary point
 - stationary gradient
 - maximum number of iterations

Public Types

- enum **Type** { **maxIter**, **statPt**, **statGd** }

Public Member Functions

- **EndCriteria** ()
default constructor
- **EndCriteria** (int maxIteration, double epsilon)
initialization constructor
- void **setPositiveOptimization** ()
- bool **checkIterationNumber** (int iteration)
- bool **checkStationaryValue** (double fold, double fnew)
- bool **checkAccuracyValue** (double f)
- bool **checkStationaryGradientNorm** (double normDiff)
- bool **checkAccuracyGradientNorm** (double norm)
- bool **operator()** (int iteration, double fold, double normgold, double fnew, double normnew, double normdiff)
test if the number of iteration is not too big and if we don't
- int **criteria** () const
return the end criteria type

Protected Attributes

- int **maxIteration_**
Maximum number of iterations.
- double **functionEpsilon_**
function and gradient epsilons
- double **gradientEpsilon_**

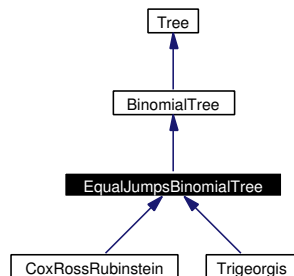
function and gradient epsilons

- int **maxIterStatPt_**
Maximun number of iterations in stationary state.
- int **statState_**
Maximun number of iterations in stationary state.
- int **endCriteria_**
- bool **positiveOptimization_**

9.140 EqualJumpsBinomialTree Class Reference

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

Inheritance diagram for EqualJumpsBinomialTree:



9.140.1 Detailed Description

Base class for equal jumps binomial tree.

Public Member Functions

- `EqualJumpsBinomialTree` (const `Handle< DiffusionProcess >` &process, Time end, Size steps)
- double `underlying` (Size i, Size index) const
- double `probability` (Size, Size, Size branch) const

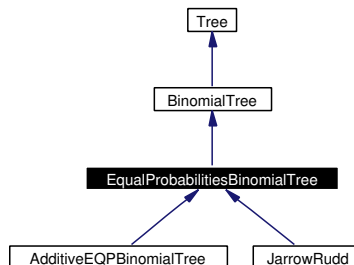
Protected Attributes

- double `dx_`
- double `pu_`
- double `pd_`

9.141 EqualProbabilitiesBinomialTree Class Reference

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

Inheritance diagram for EqualProbabilitiesBinomialTree:



9.141.1 Detailed Description

Base class for equal probabilities binomial tree.

Public Member Functions

- `EqualProbabilitiesBinomialTree` (const `Handle< DiffusionProcess >` &process, Time end, Size steps)
- double `underlying` (Size i, Size index) const
- double `probability` (Size, Size, Size) const

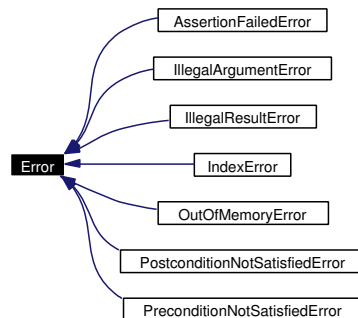
Protected Attributes

- double `up_`

9.142 Error Class Reference

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

Inheritance diagram for Error:



9.142.1 Detailed Description

Base error class.

Public Member Functions

- **Error** (const std::string &what="")
- const char * **what** () const throw ()
returns the error message.

Static Public Member Functions

- std::string **where** (const char *file, long line)

9.142.2 Constructor & Destructor Documentation

9.142.2.1 Error (const std::string & what = "") [explicit]

The explicit use of this constructor is not advised. Use the QL_FAIL macro instead.

9.143 ErrorFunction Class Reference

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

9.143.1 Detailed Description

Error function

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

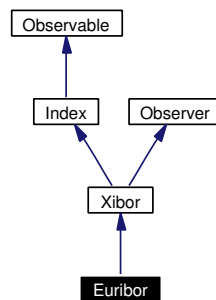
Public Member Functions

- double **operator()** (double x) const

9.144 Euribor Class Reference

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

Inheritance diagram for Euribor:



9.144.1 Detailed Description

Euribor index

Public Member Functions

- **Euribor** (int n, TimeUnit units, const **RelinkableHandle**< **TermStructure** > &h, const **Day-Counter** &dc=**Actual360**())

9.145 EuroFormatter Class Reference

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

9.145.1 Detailed Description

Formats amounts in Euro for output.

Formatting follows Euro convention (x,xxx,xxx.xx)

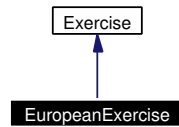
Static Public Member Functions

- `std::string toString` (double amount)

9.146 EuropeanExercise Class Reference

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

Inheritance diagram for EuropeanExercise:



9.146.1 Detailed Description

European exercise.

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

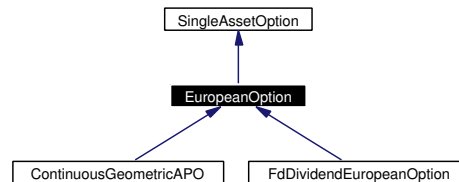
Public Member Functions

- **EuropeanExercise** (**Date** date)

9.147 EuropeanOption Class Reference

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

Inheritance diagram for EuropeanOption:



9.147.1 Detailed Description

Black-Scholes-Merton European option.

Deprecated

use `VanillaOption`(p. 624) with `EuropeanAnalyticEngine`

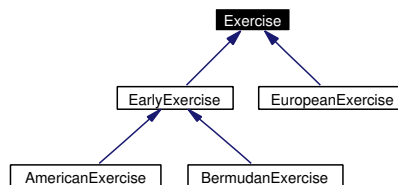
Public Member Functions

- **EuropeanOption** (Option::Type type, double underlying, double strike, Spread dividend-Yield, Rate riskFreeRate, Time residualTime, double volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const
- **Handle< SingleAssetOption > clone** () const
- void **setVolatility** (double newVolatility)
- void **setRiskFreeRate** (Rate newRate)
- void **setDividendYield** (Rate newDividendYield)
- double **beta** () const

9.148 Exercise Class Reference

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

Inheritance diagram for Exercise:



9.148.1 Detailed Description

Base exercise class.

Public Types

- enum **Type** { **American**, **Bermudan**, **European** }

Public Member Functions

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

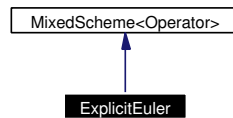
Protected Attributes

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

9.149 ExplicitEuler Class Template Reference

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

Inheritance diagram for ExplicitEuler:



9.149.1 Detailed Description

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

Forward Euler scheme for finite difference methods.

See sect. **The finite differences framework**(p. 37) for details on the method.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```

typedef ... arrayType;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
arrayType applyTo(const arrayType&);
static Operator identity(Size size);

// operator algebra
Operator operator*(double, const Operator&);
Operator operator-(const Operator&, const Operator&);

```

Todo

add Richardson extrapolation

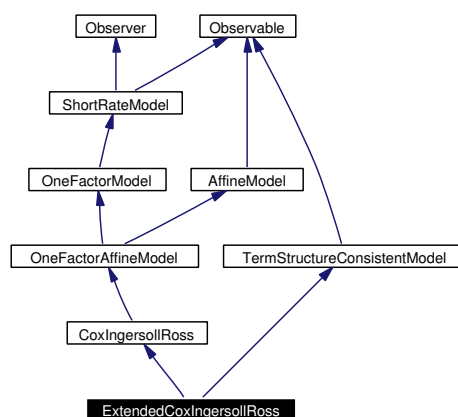
Friends

- class FiniteDifferenceModel< ExplicitEuler< Operator > >

9.150 ExtendedCoxIngersollRoss Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss:



9.150.1 Detailed Description

Extended Cox-Ingersoll-Ross model class.

This class implements the extended Cox-Ingersoll-Ross model defined by

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

Bug

This class was not tested enough to guarantee its functionality.

Public Member Functions

- **ExtendedCoxIngersollRoss** (const **RelinkableHandle**< **TermStructure** > &termStructure, double theta=0.1, double k=0.1, double sigma=0.1, double x0=0.05)
- **Handle**< **Lattice** > **tree** (const **TimeGrid** &grid) const
Return by default a trinomial recombining tree.
- **Handle**< **ShortRateDynamics** > **dynamics** () const
returns the short-rate dynamics
- double **discountBondOption** (Option::Type type, double strike, Time maturity, Time bond-Maturity) const

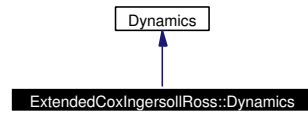
Protected Member Functions

- void **generateArguments** ()
- double **A** (Time t, Time T) const

9.151 ExtendedCoxIngersollRoss::Dynamics Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss::Dynamics:



9.151.1 Detailed Description

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

The short-rate is here

$$r_t = \varphi(t) + y_t^2$$

where $\varphi(t)$ is the deterministic time-dependent parameter used for term-structure fitting and y_t is the state variable, the square-root of a standard CIR process.

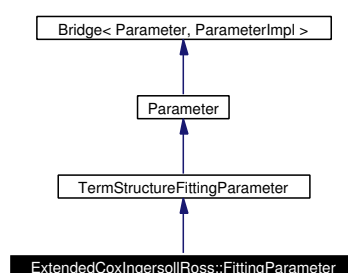
Public Member Functions

- **Dynamics** (const **Parameter** &phi, double theta, double k, double sigma, double x0)
- virtual double **variable** (Time t, Rate r) const
- virtual double **shortRate** (Time t, double y) const

9.152 ExtendedCoxIngersollRoss::FittingParameter Class Reference

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

Inheritance diagram for ExtendedCoxIngersollRoss::FittingParameter:



9.152.1 Detailed Description

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

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) - \frac{2k\theta(e^{th} - 1)}{2h + (k + h)(e^{th} - 1)} - \frac{4x_0h^2e^{th}}{(2h + (k + h)(e^{th} - 1))^1},$$

where $f(t)$ is the instantaneous forward rate at t and $h = \sqrt{k^2 + 2\sigma^2}$.

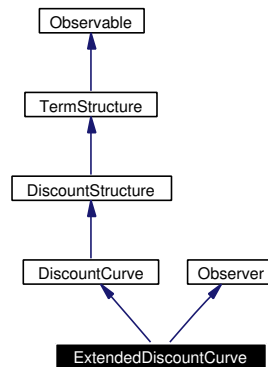
Public Member Functions

- **FittingParameter** (const **RelinkableHandle**< **TermStructure** > &termStructure, double theta, double k, double sigma, double x0)

9.153 ExtendedDiscountCurve Class Reference

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

Inheritance diagram for ExtendedDiscountCurve:



9.153.1 Detailed Description

Term structure based on loglinear interpolation of discount factors.

Loglinear interpolation guarantees piecewise constant forward rates.

Rates are assumed to be annual continuous compounding.

Public Member Functions

- **ExtendedDiscountCurve** (const **Date** &todayDate, const std::vector< **Date** > &dates, const std::vector< DiscountFactor > &dfs, const **Calendar** &calendar, const RollingConvention roll, const **DayCounter** &dayCounter=**Actual365**())
- **Calendar** **calendar** () const
- RollingConvention **roll** () const

Observer interface

- void **update** ()

Protected Member Functions

- void **calibrateNodes** () const
- **Handle**< **TermStructure** > **reversebootstrap** (int) const
- Rate **compoundForwardImpl** (Time, int, bool extrapolate=false) const
- **Handle**< **TermStructure** > **forwardCurve** (int) const

9.153.2 Member Function Documentation

9.153.2.1 void update () [virtual]

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

Implements **Observer** (p. [475](#)).

9.153.2.2 Rate compoundForwardImpl (Time, int, bool *extrapolate* = false) const [protected, virtual]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Reimplemented from **DiscountStructure** (p. [252](#)).

9.154 Factorial Class Reference

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

9.154.1 Detailed Description

Factorial numbers calculator

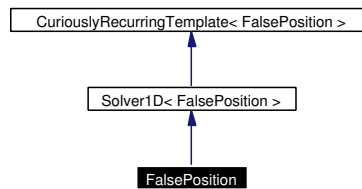
Static Public Member Functions

- double **get** (unsigned int n)
- double **ln** (unsigned int n)

9.155 FalsePosition Class Reference

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

Inheritance diagram for FalsePosition:



9.155.1 Detailed Description

False position 1-D solver.

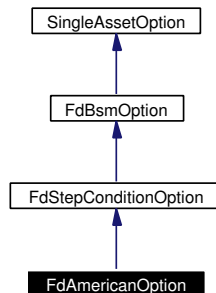
Public Member Functions

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

9.156 FdAmericanOption Class Reference

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

Inheritance diagram for FdAmericanOption:



9.156.1 Detailed Description

American option.

Public Member Functions

- **FdAmericanOption** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility, int timeSteps, int gridPoints)
- void **initializeStepCondition** () const
- **Handle< SingleAssetOption > clone** () const

9.157 FdBermudanOption Class Reference

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

9.157.1 Detailed Description

Bermudan option.

Public Member Functions

- **FdBermudanOption** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility, const std::vector< Time > &dates=std::vector< Time >(), int timeSteps=100, int gridPoints=100)
- **Handle< SingleAssetOption > clone** () const

Protected Member Functions

- void **initializeStepCondition** () const
- void **executeIntermediateStep** (int) const

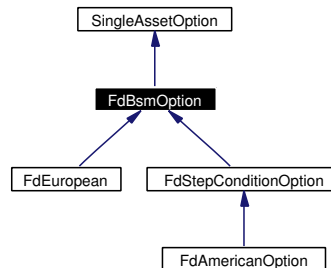
Protected Attributes

- double **extraTermInBermudan**

9.158 FdBsmOption Class Reference

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

Inheritance diagram for FdBsmOption:



9.158.1 Detailed Description

Black-Scholes-Merton option priced numerically.

Public Member Functions

- **FdBsmOption** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility, Size gridPoints)
- virtual void **calculate** () const=0
- double **value** () const
- double **delta** () const
- double **gamma** () const
- const Array & **getGrid** () const

Protected Types

- typedef **BoundaryCondition**< TridiagonalOperator > **BoundaryCondition**

Protected Member Functions

- virtual void **setGridLimits** (double center, double timeDelay) const
- virtual void **initializeGrid** () const
- virtual void **initializeInitialCondition** () const
- virtual void **initializeOperator** () const

Protected Attributes

- Size **gridPoints_**
- double **value_**
- double **delta_**
- double **gamma_**

- Array `grid_`
- BSMOperator `finiteDifferenceOperator_`
- Array `intrinsicValues_`
- `std::vector< Handle< BoundaryCondition > > BCs_`
- double `sMin_`
- double `center_`
- double `sMax_`

9.159 FdDividendAmericanOption Class Reference

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

9.159.1 Detailed Description

American option with discrete dividends.

Bug

sometimes yields negative vega when deeply in-the-money
method impliedVolatility() utterly fails

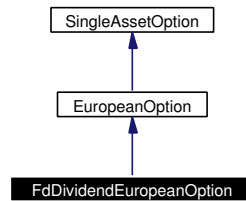
Public Member Functions

- **FdDividendAmericanOption** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility, const std::vector< double > ÷nds=std::vector< double >(), const std::vector< Time > &exdivdates=std::vector< Time >(), int timeSteps=100, int gridPoints=100)
- **Handle< SingleAssetOption > clone** () const

9.160 FdDividendEuropeanOption Class Reference

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

Inheritance diagram for FdDividendEuropeanOption:



9.160.1 Detailed Description

European option with dividends.

Public Member Functions

- **FdDividendEuropeanOption** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility, const std::vector< double > ÷nds, const std::vector< Time > &exdivdates)
- double **theta** () const
- double **rho** () const
- double **dividendRho** () const
- **Handle< SingleAssetOption > clone** () const
- double **riskless** (Rate r, std::vector< double > divs, std::vector< Time > divDates) const

9.161 FdDividendShoutOption Class Reference

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

9.161.1 Detailed Description

Shout option with dividends.

Public Member Functions

- **FdDividendShoutOption** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility, const std::vector< double > ÷nds=std::vector< double >(), const std::vector< Time > &exdivdates=std::vector< Time >(), int timeSteps=100, int gridPoints=100)
- **Handle< SingleAssetOption > clone** () const
- double **dividendRho** () const

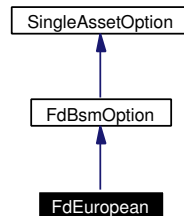
Protected Member Functions

- void **initializeStepCondition** () const

9.162 FdEuropean Class Reference

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

Inheritance diagram for FdEuropean:



9.162.1 Detailed Description

Example of European option calculated using finite differences.

Public Member Functions

- **FdEuropean** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility, Size timeSteps=200, Size gridPoints=800)
- const **Array** & **getPrices** () const
- **Handle**< **SingleAssetOption** > **clone** () const

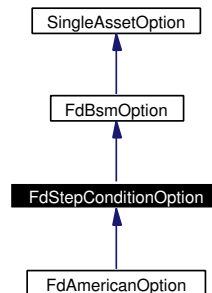
Protected Member Functions

- void **calculate** () const

9.163 FdStepConditionOption Class Reference

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

Inheritance diagram for FdStepConditionOption:



9.163.1 Detailed Description

option executing additional code at each time step

Protected Member Functions

- **FdStepConditionOption** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, Time residualTime, double volatility, int timeSteps, int gridPoints)
- void **calculate** () const
- virtual void **initializeStepCondition** () const=0

Protected Attributes

- Handle< StandardStepCondition > **stepCondition_**
- int **timeSteps_**

9.164 `filtering_iterator` Class Template Reference

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

9.164.1 Detailed Description

```
template<class Iterator, class UnaryPredicate> class QuantLib::filtering_iterator< Iterator,  
UnaryPredicate >
```

Iterator filtering undesired data.

This iterator advances an underlying iterator returning only those data satisfying a given condition.

Public Member Functions

- **`filtering_iterator`** (const `Iterator` &, const `UnaryPredicate` &, const `Iterator` &beforeBegin, const `Iterator` &end)

Dereferencing

- reference **`operator *`** () const
- pointer **`operator →`** () const

Increment and decrement

- **`filtering_iterator`** & **`operator++`** ()
- **`filtering_iterator`** **`operator++`** (int)
- **`filtering_iterator`** & **`operator--`** ()
- **`filtering_iterator`** **`operator--`** (int)

Comparisons

- bool **`operator==`** (const **`filtering_iterator`**< `Iterator`, `UnaryPredicate` > &)
- bool **`operator!=`** (const **`filtering_iterator`**< `Iterator`, `UnaryPredicate` > &)

Public Attributes

- typedef< `Iterator` >::pointer **`pointer`**
- typedef< `Iterator` >::reference **`reference`**

Related Functions

(Note that these are not member functions.)

- **`filtering_iterator`**< `Iterator`, `UnaryPredicate` > **`make_filtering_iterator`** (`Iterator` it, `UnaryPredicate` p, `Iterator` beforeBegin, `Iterator` end)

helper function to create filtering iterators

9.165 FiniteDifferenceModel Class Template Reference

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

9.165.1 Detailed Description

```
template<class Evolver> class QuantLib::FiniteDifferenceModel< Evolver >
```

Generic finite difference model.

See sect. [The finite differences framework](#)(p. 37)

Public Types

- typedef Evolver::arrayType **arrayType**
- typedef Evolver::operatorType **operatorType**
- typedef **BoundaryCondition**< operatorType > **bcType**
- typedef **StepCondition**< arrayType > **conditionType**

Public Member Functions

- **FiniteDifferenceModel** (const operatorType &L, const std::vector< **Handle**< bcType > &bcs, const std::vector< Time > &stoppingTimes=std::vector< Time >())
- **FiniteDifferenceModel** (const Evolver &evolver, const std::vector< Time > &stoppingTimes=std::vector< Time >())
- void **rollback** (arrayType &a, Time from, Time to, Size steps, const **Handle**< **conditionType** > &condition=**Handle**< **conditionType** >())
- const Evolver & **evolver** () const

9.165.2 Member Function Documentation

9.165.2.1 void **rollback** (arrayType & *a*, Time *from*, Time *to*, Size *steps*, const **Handle**< **conditionType** > & *condition* = **Handle**< **conditionType** >())

solves the problem between the given times, possibly applying a condition at every step.

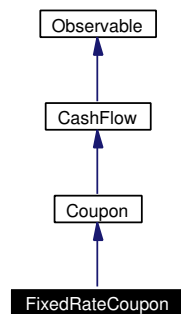
Warning:

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

9.166 FixedRateCoupon Class Reference

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

Inheritance diagram for FixedRateCoupon:



9.166.1 Detailed Description

Coupon paying a fixed interest rate

Public Member Functions

- **FixedRateCoupon** (double nominal, const **Date** &paymentDate, Rate rate, const **DayCounter** &dayCounter, const **Date** &startDate, const **Date** &endDate, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())

CashFlow interface

- double **amount** () const
returns the amount of the cash flow

Coupon interface

- **DayCounter dayCounter** () const
day counter for accrual calculation
- double **accruedAmount** (const **Date** &) const
accrued amount at the given date

Inspectors

- Rate **rate** () const

Visitability

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

9.166.2 Member Function Documentation

9.166.2.1 `double amount () const` [virtual]

returns the amount of the cash flow

Note:

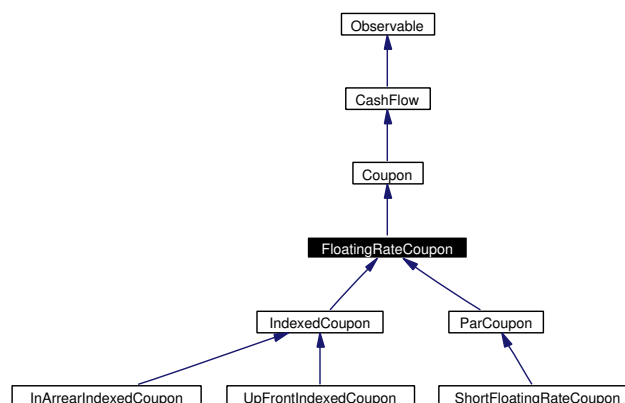
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements **CashFlow** (p. [200](#)).

9.167 FloatingRateCoupon Class Reference

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

Inheritance diagram for FloatingRateCoupon:



9.167.1 Detailed Description

Coupon at par on a term structure

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **FloatingRateCoupon** (double nominal, const **Date** &paymentDate, const **Date** &startDate, const **Date** &endDate, int fixingDays, Spread spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())

Coupon interface

- double **accruedAmount** (const **Date** &) const
accrued amount at the given date

Inspectors

- int **fixingDays** () const
- virtual Spread **spread** () const
- virtual Rate **fixing** () const=0
- virtual **Date** **fixingDate** () const=0

Visitability

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

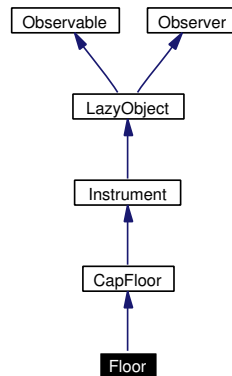
Protected Attributes

- int `fixingDays_`
- Spread `spread_`

9.168 Floor Class Reference

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

Inheritance diagram for Floor:



9.168.1 Detailed Description

Concrete floor class.

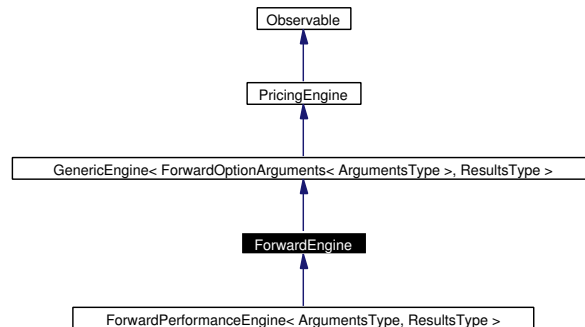
Public Member Functions

- **Floor** (const std::vector< **Handle**< **CashFlow** > > &floatingLeg, const std::vector< Rate > &exerciseRates, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)

9.169 ForwardEngine Class Template Reference

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

Inheritance diagram for ForwardEngine:



9.169.1 Detailed Description

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

Forward engine base class.

Public Member Functions

- **ForwardEngine** (const **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > &)
- void **setOriginalArguments** () const
- void **calculate** () const
- void **getOriginalResults** () const

Protected Attributes

- **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

9.170 ForwardOptionArguments Class Template Reference

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

9.170.1 Detailed Description

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

Arguments for forward (strike-resetting) option calculation

Public Member Functions

- void **validate** () const

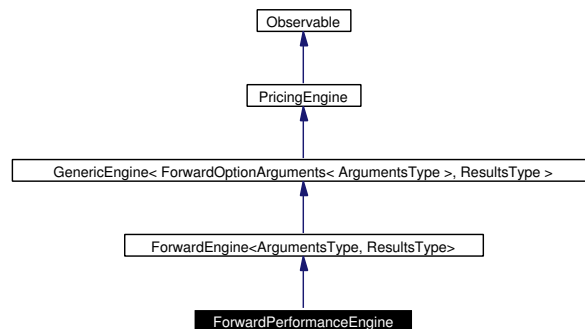
Public Attributes

- double **moneyness**
- Date **resetDate**

9.171 ForwardPerformanceEngine Class Template Reference

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

Inheritance diagram for ForwardPerformanceEngine:



9.171.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::ForwardPerformance-  
Engine< ArgumentsType, ResultsType >
```

Forward performance engine.

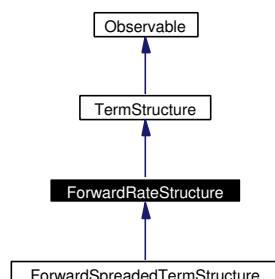
Public Member Functions

- **ForwardPerformanceEngine** (const **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > &)
- void **calculate** () const
- void **getOriginalResults** () const

9.172 ForwardRateStructure Class Reference

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

Inheritance diagram for ForwardRateStructure:



9.172.1 Detailed Description

Forward rate term structure.

This abstract class acts as an adapter to [TermStructure](#)(p. 594) allowing the programmer to implement only the `forwardImpl(const Date&, bool)` method in derived classes.

Rates are assumed to be annual continuous compounding.

Protected Member Functions

- Rate **zeroYieldImpl** (Time, bool *extrapolate*=false) const
- DiscountFactor **discountImpl** (Time, bool *extrapolate*=false) const
- Rate **compoundForwardImpl** (Time, int, bool *extrapolate*=false) const

9.172.2 Member Function Documentation

9.172.2.1 Rate **zeroYieldImpl** (Time, bool *extrapolate* = false) const [protected, virtual]

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

Warning:

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

Implements [TermStructure](#) (p. 595).

Reimplemented in [ForwardSpreadedTermStructure](#) (p. 314).

9.172.2.2 DiscountFactor **discountImpl** (Time, bool *extrapolate* = false) const [protected, virtual]

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

Implements [TermStructure](#) (p. 595).

9.172.2.3 Rate compoundForwardImpl (Time, int, bool *extrapolate* = false) const [protected, virtual]

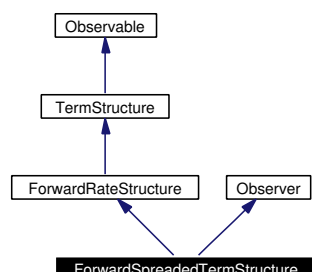
Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Implements **TermStructure** (p. [595](#)).

9.173 ForwardSpreadedTermStructure Class Reference

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

Inheritance diagram for ForwardSpreadedTermStructure:



9.173.1 Detailed Description

Term structure with added spread on the instantaneous forward rate.

Note:

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

Public Member Functions

- **ForwardSpreadedTermStructure** (const **RelinkableHandle**< **TermStructure** > &, const **RelinkableHandle**< **Quote** > &spread)

TermStructure interface

- **DayCounter** **dayCounter** () const
the day counter used for date/time conversion
- **Date** **todaysDate** () const
today's date
- **Date** **referenceDate** () const
the reference date, i.e., the date at which discount = 1
- **Date** **maxDate** () const
the latest date for which the curve can return rates
- **Time** **maxTime** () const
the latest time for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- Rate **forwardImpl** (Time, bool extrapolate=false) const
returns the spreaded forward rate
- Rate **zeroYieldImpl** (Time, bool extrapolate=false) const
returns the spreaded zero yield rate

9.173.2 Member Function Documentation

9.173.2.1 void update () [virtual]

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

Implements **Observer** (p. [475](#)).

9.173.2.2 Rate zeroYieldImpl (Time, bool *extrapolate* = false) const [protected, virtual]

returns the spreaded zero yield rate

Warning:

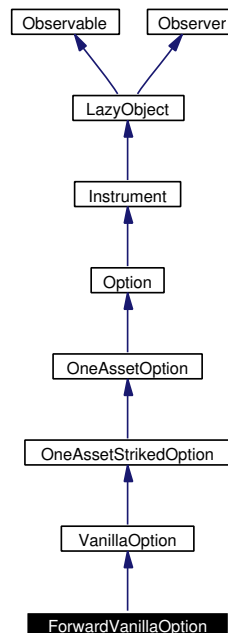
This method must disappear should the spread become a curve

Reimplemented from **ForwardRateStructure** (p. [311](#)).

9.174 ForwardVanillaOption Class Reference

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

Inheritance diagram for ForwardVanillaOption:



9.174.1 Detailed Description

Forward version of a vanilla option.

Public Types

- typedef **ForwardOptionArguments**< VanillaOption::arguments > **arguments**
- typedef VanillaOption::results **results**

Public Member Functions

- **ForwardVanillaOption** (double moneyness, **Date** resetDate, const **Handle**< BlackScholes-StochasticProcess > &stochProc, const **Handle**< **StrikedTypePayoff** > &payoff, const **Handle**< **Exercise** > &exercise, const **Handle**< **PricingEngine** > &engine)
- void **setupArguments** (**Arguments** *) const

Protected Member Functions

- void **performCalculations** () const

9.174.2 Member Function Documentation

9.174.2.1 `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** (p. [482](#)).

9.174.2.2 `void performCalculations () const` [protected, virtual]

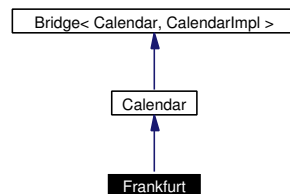
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **VanillaOption** (p. [624](#)).

9.175 Frankfurt Class Reference

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

Inheritance diagram for Frankfurt:



9.175.1 Detailed Description

Frankfurt calendar

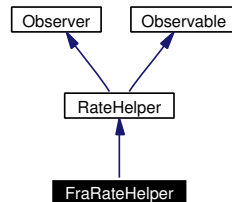
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Ascension Thursday
- Whit Monday
- Corpus Christi
- Labour Day, May 1st
- National Day, October 3rd
- Christmas Eve, December 24th
- Christmas, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31st

9.176 FraRateHelper Class Reference

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

Inheritance diagram for FraRateHelper:



9.176.1 Detailed Description

Forward rate agreement.

Warning:

This class assumes that the reference date does not change between calls of **setTermStructure()**(p. 318).

Todo

convexity adjustment should be implemented.

Public Member Functions

- **FraRateHelper** (const **RelinkableHandle**< **Quote** > &rate, int monthsToStart, int monthsToEnd, int settlementDays, const **Calendar** &calendar, RollingConvention convention, const **DayCounter** &dayCounter)
- **FraRateHelper** (double rate, int monthsToStart, int monthsToEnd, int settlementDays, const **Calendar** &calendar, RollingConvention convention, const **DayCounter** &dayCounter)
- double **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- void **setTermStructure** (**TermStructure** *)
sets the term structure to be used for pricing
- **Date maturity** () const
maturity date

9.176.2 Member Function Documentation

9.176.2.1 void setTermStructure (TermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:

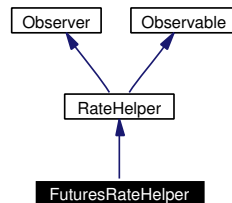
Being a pointer and not a **Handle**(p. 339), the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from **RateHelper** (p. 537).

9.177 FuturesRateHelper Class Reference

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

Inheritance diagram for FuturesRateHelper:



9.177.1 Detailed Description

Interest-rate futures.

Warning:

This class assumes that the reference date does not change between calls of **setTermStructure()**(p. 537).

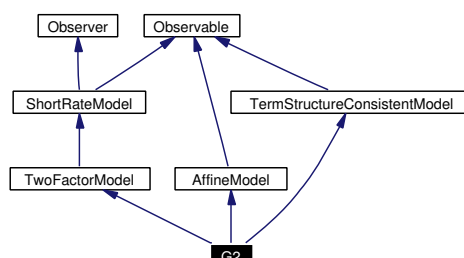
Public Member Functions

- **FuturesRateHelper** (const **RelinkableHandle**< **Quote** > &price, const **Date** &ImmDate, int nMonths, const **Calendar** &calendar, RollingConvention convention, const **DayCounter** &dayCounter)
- **FuturesRateHelper** (const **RelinkableHandle**< **Quote** > &price, const **Date** &ImmDate, const **Date** &MatDate, const **Calendar** &calendar, RollingConvention convention, const **DayCounter** &dayCounter)
- **FuturesRateHelper** (double price, const **Date** &ImmDate, int nMonths, const **Calendar** &calendar, RollingConvention convention, const **DayCounter** &dayCounter)
- double **impliedQuote** () const
- DiscountFactor **discountGuess** () const
- **Date** **maturity** () const
maturity date

9.178 G2 Class Reference

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

Inheritance diagram for G2:



9.178.1 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.

Public Member Functions

- **G2** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a=0.1, double sigma=0.01, double b=0.1, double eta=0.01, double rho=0.9)
- **Handle**< **ShortRateDynamics** > **dynamics** () const
Returns the short-rate dynamics.
- double **discountBondOption** (Option::Type type, double strike, Time maturity, Time bond-Maturity) const
- double **swaption** (const **Swaption::arguments** &arguments) const
- DiscountFactor **discount** (Time t) const
Implied discount curve.

Protected Member Functions

- void **generateArguments** ()
- double **A** (Time t, Time T) const
- double **B** (double x, Time t) const

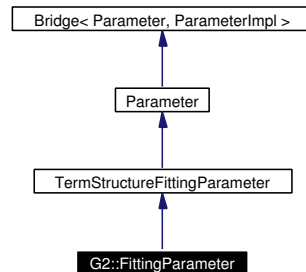
Friends

- class `SwaptionPricingFunction`

9.179 G2::FittingParameter Class Reference

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

Inheritance diagram for G2::FittingParameter:



9.179.1 Detailed Description

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

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) + \frac{1}{2} \left(\frac{\sigma(1 - e^{-at})}{a} \right)^2 + \frac{1}{2} \left(\frac{\eta(1 - e^{-bt})}{b} \right)^2 + \rho \frac{\sigma(1 - e^{-at})}{a} \frac{\eta(1 - e^{-bt})}{b},$$

where $f(t)$ is the instantaneous forward rate at t .

Public Member Functions

- **FittingParameter** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a, double sigma, double b, double eta, double rho)

9.180 GammaFunction Class Reference

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

9.180.1 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

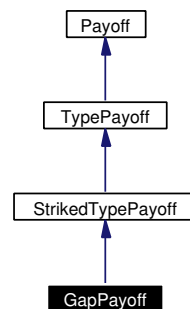
Public Member Functions

- double **logValue** (double x) const

9.181 GapPayoff Class Reference

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

Inheritance diagram for GapPayoff:



9.181.1 Detailed Description

Binary gap payoff.

Public Member Functions

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

9.182 GaussianStatistics Class Template Reference

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

9.182.1 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

- double **gaussianDownsideVariance** () const
- double **gaussianDownsideDeviation** () const
- double **gaussianRegret** (double target) const
- double **gaussianPercentile** (double percentile) const
- double **gaussianPotentialUpside** (double percentile) const
gaussian-assumption Potential-Upside at a given percentile
- double **gaussianValueAtRisk** (double percentile) const
gaussian-assumption Value-At-Risk at a given percentile
- double **gaussianExpectedShortfall** (double percentile) const
gaussian-assumption Expected Shortfall at a given percentile
- double **gaussianShortfall** (double target) const
gaussian-assumption Shortfall (observations below target)
- double **gaussianAverageShortfall** (double target) const
*gaussian-assumption **Average**(p. 132) Shortfall (averaged shortfallness)*

9.182.2 Member Function Documentation

9.182.2.1 double gaussianDownsideVariance () const

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^N \theta \times x_i^2}{\sum_{i=1}^N w_i}$$

, where $\theta = 0$ if $x > 0$ and $\theta = 1$ if $x < 0$

9.182.2.2 double gaussianDownsideDeviation () const

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

9.182.2.3 double gaussianRegret (double *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)

9.182.2.4 double gaussianPercentile (double *percentile*) const

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

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

9.182.2.5 double gaussianPotentialUpside (double *percentile*) const

gaussian-assumption Potential-Upside at a given percentile

Precondition:

percentile must be in range [90%-100%)

9.182.2.6 double gaussianValueAtRisk (double *percentile*) const

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

Precondition:

percentile must be in range [90%-100%)

9.182.2.7 double gaussianExpectedShortfall (double *percentile*) const

gaussian-assumption Expected Shortfall at a given percentile

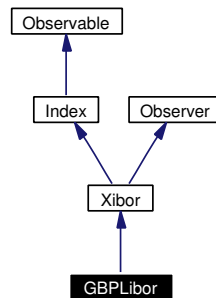
Precondition:

percentile must be in range 90%-100%

9.183 GBPLibor Class Reference

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

Inheritance diagram for GBPLibor:



9.183.1 Detailed Description

GBP Libor index

Public Member Functions

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

9.184 GeneralStatistics Class Reference

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

9.184.1 Detailed Description

Statistics tool.

This class accumulates a set of data and returns their statistics (e.g: mean, variance, skewness, kurtosis, error estimation, percentile, etc.) based on the empirical distribution (no gaussian assumption)

It doesn't suffer the numerical instability problem of **IncrementalStatistics**(p. 360). The downside is that it stores all samples, thus increasing the memory requirements.

Public Member Functions

Inspectors

- Size **samples** () const
number of samples collected
- const std::vector< std::pair< double, double > > & **data** () const
collected data
- double **weightSum** () const
sum of data weights
- double **mean** () const
- double **variance** () const
- double **standardDeviation** () const
- double **errorEstimate** () const
- double **skewness** () const
- double **kurtosis** () const
- double **min** () const
- double **max** () const
- template<class Func, class Predicate> std::pair< double, Size > **expectationValue** (const Func &f, const Predicate &inRange) const
- double **percentile** (double y) const
- double **topPercentile** (double y) const

Modifiers

- void **add** (double value, double weight=1.0)
adds a datum to the set, possibly with a weight
- template<class DataIterator> void **addSequence** (DataIterator begin, DataIterator end)
adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator> void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)
adds a sequence of data to the set, each with its weight
- void **reset** ()

resets the data to a null set

- void **sort** () const
sort the data set in increasing order

9.184.2 Member Function Documentation

9.184.2.1 double mean () const

returns the mean, defined as

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

9.184.2.2 double variance () const

returns the variance, defined as

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

9.184.2.3 double standardDeviation () const

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

9.184.2.4 double errorEstimate () const

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

9.184.2.5 double 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.

9.184.2.6 double 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.

9.184.2.7 double min () const

returns the minimum sample value

9.184.2.8 double max () const

returns the maximum sample value

9.184.2.9 std::pair<double,Size> expectationValue (const Func & *f*, const Predicate & *inRange*) const

Expectation value of a function f on a given range \mathcal{R} , i.e.,

$$E[f | \mathcal{R}] = \frac{\sum_{x_i \in \mathcal{R}} f(x_i) w_i}{\sum_{x_i \in \mathcal{R}} w_i}.$$

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.

9.184.2.10 double percentile (double *y*) const

y -th percentile, defined as the value \bar{x} such that

$$y = \frac{\sum_{x_i < \bar{x}} w_i}{\sum_i w_i}$$

Precondition:

y must be in the range $(0 - 1]$.

9.184.2.11 double topPercentile (double *y*) const

y -th top percentile, defined as the value \bar{x} such that

$$y = \frac{\sum_{x_i > \bar{x}} w_i}{\sum_i w_i}$$

Precondition:

y must be in the range $(0 - 1]$.

9.184.2.12 void add (double *value*, double *weight* = 1.0)

adds a datum to the set, possibly with a weight

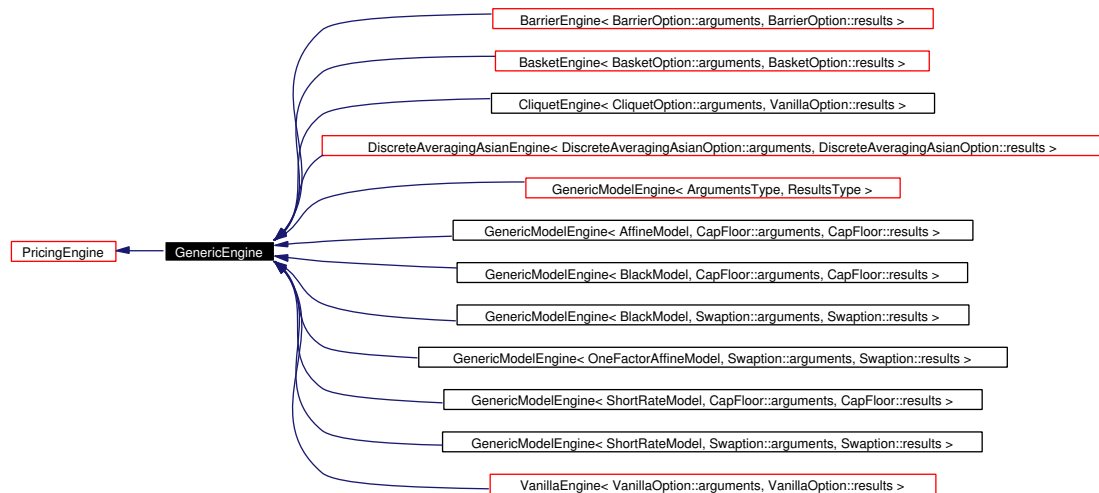
Precondition:

weights must be positive or null

9.185 GenericEngine Class Template Reference

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

Inheritance diagram for GenericEngine:



9.185.1 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`

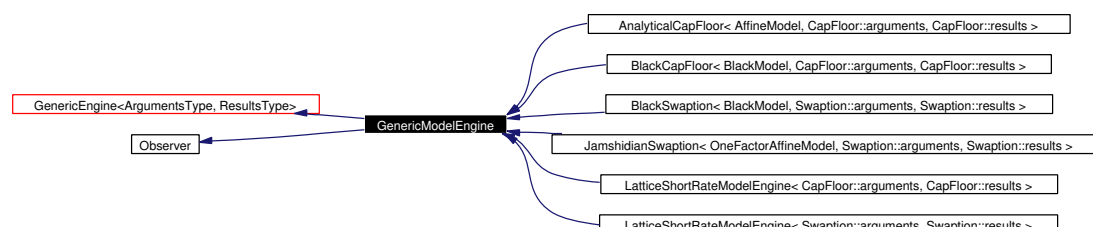
Protected Attributes

- `ArgumentsType arguments_`
- `ResultsType results_`

9.186 GenericModelEngine Class Template Reference

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

Inheritance diagram for GenericModelEngine:



9.186.1 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 **Handle**< ModelType > &model)
- void **validateArguments** () const
- void **setModel** (const **Handle**< ModelType > &model)
- virtual void **update** ()

Protected Attributes

- **Handle**< ModelType > **model_**

9.186.2 Member Function Documentation

9.186.2.1 virtual void update () [virtual]

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

Implements **Observer** (p. 475).

Reimplemented in **LatticeShortRateModelEngine** (p. 389), **LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >** (p. 389), and **LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >** (p. 389).

9.187 GenericRiskStatistics Class Template Reference

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

9.187.1 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

- double **semiVariance** () const
- double **semiDeviation** () const
- double **downsideVariance** () const
- double **downsideDeviation** () const
- double **regret** (double target) const
- double **potentialUpside** (double percentile) const
potential upside (the reciprocal of VAR) at a given percentile
- double **valueAtRisk** (double percentile) const
value-at-risk at a given percentile
- double **expectedShortfall** (double percentile) const
expected shortfall at a given percentile
- double **shortfall** (double target) const
- double **averageShortfall** (double target) const

9.187.2 Member Function Documentation

9.187.2.1 double 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).

9.187.2.2 double semiDeviation () const

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

9.187.2.3 double downsideVariance () const

returns the variance of observations below 0.0,

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

9.187.2.4 double downsideDeviation () const

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

9.187.2.5 double regret (double *target*) const

returns the variance of observations below target,

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

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

9.187.2.6 double potentialUpside (double *centile*) const

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

Precondition:

percentile must be in range [90%-100%)

9.187.2.7 double valueAtRisk (double *centile*) const

value-at-risk at a given percentile

Precondition:

percentile must be in range [90%-100%)

9.187.2.8 double expectedShortfall (double *percentile*) const

expected shortfall at a given percentile

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

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

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)

9.187.2.9 double shortfall (double *target*) const

probability of missing the given target, defined as

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

where

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

9.187.2.10 double averageShortfall (double *target*) const

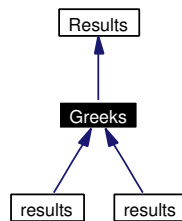
averaged shortfallness, defined as

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

9.188 Greeks Class Reference

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

Inheritance diagram for Greeks:



9.188.1 Detailed Description

additional option results

Public Member Functions

- void **reset** ()

Public Attributes

- double **delta**
- double **gamma**
- double **theta**
- double **vega**
- double **rho**
- double **dividendRho**

9.189 HaltonRsg Class Reference

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

9.189.1 Detailed Description

Halton low-discrepancy sequence generator.

Halton algorithm for low-discrepancy sequence. For more details see chapter 8, paragraph 2 of "Monte Carlo Methods in Finance", by Peter Jäckel

Public Types

- typedef **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

9.190 Handle Class Template Reference

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

9.190.1 Detailed Description

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

Reference-counted pointer.

This class acts as a proxy to a pointer contained in it. Such pointer is owned by the handle, i.e., the handle will be responsible for its deletion, unless explicitly stated by the programmer.

A count of the references to the contained pointer is incremented every time a handle is copied, and decremented every time a handle is deleted or goes out of scope. When the handle owns the pointer, this mechanism ensures on one hand, that the pointer will not be deallocated as long as a handle refers to it, and on the other hand, that it will be deallocated when no more handles do.

Note:

The implementation of this class was originally taken from "The C++ Programming Language", 3rd ed., B.Stroustrup, Addison-Wesley, 1997.

Warning:

This mechanism will break and result in untimely deallocation of the pointer (and very possible death of your executable) if two handles are explicitly initialized with the same pointer, as in

```
SomeObj* so = new SomeObj;
Handle<SomeObj> h1(so);
Handle<SomeObj> h2 = h1;    // this is safe.
Handle<SomeObj> h3(so);    // this is definitely not.
```

It is good practice to create the pointer and immediately pass it to the handle, as in

```
Handle<SomeObj> h1(new SomeObj);    // this is as safe as can be.
```

When the programmer keeps the ownership of the pointer, as explicitly declared in

```
SomeObj so;
Handle<SomeObj> h(&so,false);
```

it is responsibility of the programmer to make sure that the object remain in scope as long as there are handles pointing to it. Also, the programmer must explicitly delete the object if required.

Public Member Functions

constructors, destructor, and assignment

- **Handle** (T *ptr=0, bool owns=true)
Constructor taking a pointer.
- **template<class U> Handle** (const **Handle**< U > &from)
- **Handle** (const **Handle** &from)
- **template<class U> Handle & operator=** (const **Handle**< U > &from)
- **Handle & operator=** (const **Handle** &from)

Dereferencing

- `T & operator * () const`
- `T * operator → () const`

Inspectors

- `bool isNull () const`
Checks if the contained pointer is actually allocated.
- `bool operator== (const Handle< T > &) const`
Checks if the two handles point to the same object.
- `bool operator!= (const Handle< T > &) const`

Friends

- class `HandleCopier`

9.190.2 Constructor & Destructor Documentation

9.190.2.1 `Handle (T * ptr = 0, bool owns = true)` [explicit]

Constructor taking a pointer.

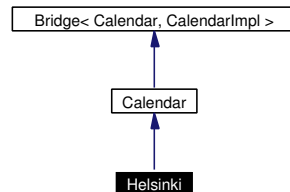
If `owns` is set to `true` (the default), the handle will be responsible for the deletion of the pointer. If it is set to `false`, the programmer must make sure that the pointed object remains in scope for the lifetime of the handle and its copies. Destruction of the object is also responsibility of the programmer.

It is advised that handles be used with `owns = false` only in a controlled and self-contained environment. Such a case happens when an object needs to pass a handle to itself to inner classes or bootstrappers - i.e., contained or temporary objects whose lifetime is guaranteed not to last more than the lifetime of the object.

9.191 Helsinki Class Reference

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

Inheritance diagram for Helsinki:



9.191.1 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

9.192 History Class Reference

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

9.192.1 Detailed Description

Container for historical data.

This class acts as a generic repository for a set of historical data. Single data can be accessed through their date, while sets of consecutive data can be accessed through iterators.

A history can contain null data, which can either be returned or skipped according to the chosen iterator type.

Example: [uses of history iterators](#) (p. ??)

Public Types

- typedef **filtering_iterator**< **const_iterator**, DataValidator > **const_valid_iterator**
bidirectional iterator on non-null history entries
- typedef std::vector< double >::**const_iterator** **const_data_iterator**
random access iterator on historical data
- typedef **filtering_iterator**< **const_data_iterator**, DataValidator > **const_valid_data_iterator**
bidirectional iterator on non-null historical data

Public Member Functions

- **History** ()
- template<class Iterator> **History** (const **Date** &firstDate, const **Date** &lastDate, Iterator begin, Iterator end)
- **History** (const **Date** &firstDate, const std::vector< double > &values)
- **History** (const **Date** &firstDate, const **Date** &lastDate, const std::vector< double > &values)
- **History** (const std::vector< **Date** > &dates, const std::vector< double > &values)

Inspectors

- const **Date** & **firstDate** () const
returns the first date for which a historical datum exists
- const **Date** & **lastDate** () const
returns the last date for which a historical datum exists
- int **size** () const
returns the number of historical data including null ones

Historical data access

- **double operator[] (const Date &) const**
returns the (possibly null) datum corresponding to the given date

Iterator access

Four different types of iterators are provided, namely, *const_iterator*, *const_valid_iterator*, *const_data_iterator*, and *const_valid_data_iterator*.

const_iterator and *const_valid_iterator* point to an *Entry* structure, the difference being that the latter only iterates over valid entries - i.e., entries whose data are not null. The same difference exists between *const_data_iterator* and *const_valid_data_iterator* which point directly to historical values without reference to the date they are associated to.

- **const_iterator begin () const**
- **const_iterator end () const**
- **const_iterator iterator (const Date &d) const**
- **const_valid_iterator vbegin () const**
- **const_valid_iterator vend () const**
- **const_valid_iterator valid_iterator (const Date &d) const**
- **const_data_iterator dbegin () const**
- **const_data_iterator dend () const**
- **const_data_iterator data_iterator (const Date &d) const**
- **const_valid_data_iterator vdbegin () const**
- **const_valid_data_iterator vdend () const**
- **const_valid_data_iterator valid_data_iterator (const Date &d) const**

9.192.2 Constructor & Destructor Documentation

9.192.2.1 History ()

Default constructor

9.192.2.2 History (const Date & *firstDate*, const Date & *lastDate*, Iterator *begin*, Iterator *end*)

This constructor initializes the history with the given set of values, corresponding to the date range between *firstDate* and *lastDate* included.

Precondition:

begin-end must equal the number of days from *firstDate* to *lastDate* included.

9.192.2.3 History (const Date & *firstDate*, const Date & *lastDate*, const std::vector< double > & *values*)

This constructor initializes the history with the given set of values, corresponding to the date range between *firstDate* and *lastDate* included.

Precondition:

The size of *values* must equal the number of days from *firstDate* to *lastDate* included.

9.192.2.4 History (const std::vector< Date > & *dates*, const std::vector< double > & *values*)

This constructor initializes the history with the given set of values, corresponding each to the element with the same index in the given set of dates. The whole date range between *dates*[0] and *dates*[N-1] will be automatically filled by inserting null values where a date is missing from the given set.

Precondition:

dates must be sorted.

There can be no pairs (*dates*[i],*values*[i]) and (*dates*[j],*values*[j]) such that *dates*[i] == *dates*[j] && *values*[i] != *values*[j]. Pairs with *dates*[i] == *dates*[j] && *values*[i] == *values*[j] are allowed; the duplicated entries will be discarded.

The size of *values* must equal the number of days from *firstDate* to *lastDate* included.

9.193 History::const_iterator Class Reference

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

9.193.1 Detailed Description

random access iterator on history entries

Public Types

- typedef Entry **value_type**
- typedef int **difference_type**
- typedef const Entry * **pointer**
- typedef const Entry & **reference**

Public Member Functions

Dereferencing

- **reference operator *** () const
- **pointer operator →** () const

Random access

- **value_type operator[]** (difference_type i) const

Increment and decrement

- **const_iterator & operator++** ()
- **const_iterator operator++** (int)
- **const_iterator & operator--** ()
- **const_iterator operator--** (int)
- **const_iterator & operator+=** (difference_type i)
- **const_iterator & operator-=** (difference_type i)
- **const_iterator operator+** (difference_type i)
- **const_iterator operator-** (difference_type i)

Difference

- **difference_type operator-** (const **const_iterator** &i)

Comparisons

- **bool operator==** (const **const_iterator** &i)
- **bool operator!=** (const **const_iterator** &i)
- **bool operator<** (const **const_iterator** &i)
- **bool operator>** (const **const_iterator** &i)
- **bool operator<=** (const **const_iterator** &i)
- **bool operator>=** (const **const_iterator** &i)

Friends

- class **History**

9.194 History::Entry Class Reference

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

9.194.1 Detailed Description

single datum in history

Public Member Functions

- const **Date** & **date** () const
- double **value** () const

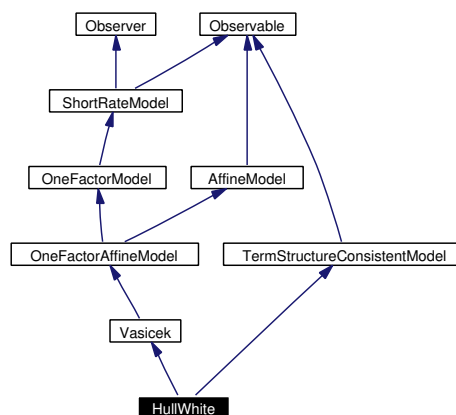
Friends

- class **const_iterator**

9.195 HullWhite Class Reference

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

Inheritance diagram for HullWhite:



9.195.1 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.

Public Member Functions

- **HullWhite** (const **RelinkableHandle**< **TermStructure** > &termStructure, double a=0.1, double sigma=0.01)
- **Handle**< **Lattice** > **tree** (const **TimeGrid** &grid) const
Return by default a trinomial recombining tree.
- **Handle**< **ShortRateDynamics** > **dynamics** () const
returns the short-rate dynamics
- double **discountBondOption** (Option::Type type, double strike, Time maturity, Time bond-Maturity) const

Protected Member Functions

- void **generateArguments** ()
- double **A** (Time t, Time T) const

9.196 HullWhite::Dynamics Class Reference

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

9.196.1 Detailed Description

Short-rate dynamics in the Hull-White model.

The short-rate is here

$$r_t = \varphi(t) + x_t$$

where $\varphi(t)$ is the deterministic time-dependent parameter used for term-structure fitting and x_t is the state variable following an Ornstein-Uhlenbeck process.

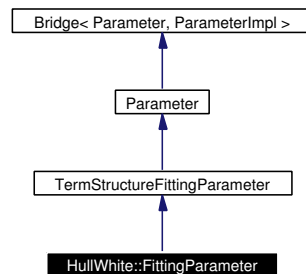
Public Member Functions

- **Dynamics** (const **Parameter** &fitting, double a, double sigma)
- double **variable** (Time t, Rate r) const
- double **shortRate** (Time t, double x) const

9.197 HullWhite::FittingParameter Class Reference

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

Inheritance diagram for HullWhite::FittingParameter:



9.197.1 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 **RelinkableHandle**< **TermStructure** > &termStructure, double a, double sigma)

9.198 ICGaussianRng Class Template Reference

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

9.198.1 Detailed Description

```
template<class RNG, class I> class QuantLib::ICGaussianRng< RNG, I >
```

Inverse cumulative Gaussian random number generator.

It uses a uniform deviate in (0, 1) as the source of cumulative normal distribution values. Then an inverse cumulative normal distribution is used as it is approximately a Gaussian deviate with average 0.0 and standard deviation 1.0.

The uniform deviate is supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

The inverse cumulative normal distribution is supplied by I.

Class I must implement the following interface:

```
I::I();  
double I::operator() const;
```

Public Types

- typedef `Sample< double > sample_type`

Public Member Functions

- `ICGaussianRng (const RNG &uniformGenerator)`
- `ICGaussianRng (long seed=0)`
- `sample_type next () const`

returns next sample from the Gaussian distribution

9.198.2 Constructor & Destructor Documentation

9.198.2.1 ICGaussianRng (long seed = 0) [explicit]

Deprecated

initialize with a random number generator instead.

9.199 ICGaussianRsg Class Template Reference

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

9.199.1 Detailed Description

template<class USG, class I> class QuantLib::ICGaussianRsg< USG, I >

Inverse cumulative Gaussian random sequence generator.

It uses a sequence of uniform deviate in (0, 1) as the source of cumulative normal distribution values. Then an inverse cumulative normal distribution is used as it is approximately a Gaussian deviate with average 0.0 and standard deviation 1.0.

The uniform deviate sequence is supplied by USG.

Class USG must implement the following interface:

```
USG::sample_type USG::next() const;  
Size USG::dimension() const;
```

The inverse cumulative normal distribution is supplied by I.

Class I must implement the following interface:

```
I::I();  
double I::operator() const;
```

Public Types

- typedef **Sample< Array > sample_type**

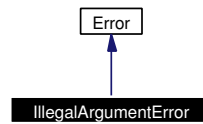
Public Member Functions

- **ICGaussianRsg** (const USG &uniformSequenceGenerator)
- const sample_type & **nextSequence** () const
returns next sample from the Gaussian distribution
- const sample_type & **lastSequence** () const
- Size **dimension** () const

9.200 IllegalArgumentError Class Reference

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

Inheritance diagram for IllegalArgumentError:



9.200.1 Detailed Description

Specialized error.

Thrown upon passing an argument with an illegal value.

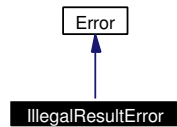
Public Member Functions

- `IllegalArgumentError` (const std::string &what="")

9.201 `IllegalResultError` Class Reference

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

Inheritance diagram for `IllegalResultError`:



9.201.1 Detailed Description

Specialized error.

Thrown upon obtaining a result outside the allowed range.

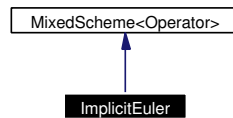
Public Member Functions

- `IllegalResultError` (const std::string &what="")

9.202 ImplicitEuler Class Template Reference

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

Inheritance diagram for ImplicitEuler:



9.202.1 Detailed Description

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

Backward Euler scheme for finite difference methods.

See sect. **The finite differences framework**(p. 37) for details on the method.

In this implementation, the passed operator must be derived from either TimeConstantOperator or TimeDependentOperator. Also, it must implement at least the following interface:

```

typedef ... arrayType;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
arrayType solveFor(const arrayType&);
static Operator identity(Size size);

// operator algebra
Operator operator*(double, const Operator&);
Operator operator+(const Operator&, const Operator&);

```

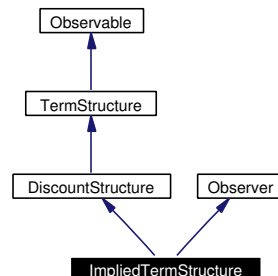
Friends

- class FiniteDifferenceModel< ImplicitEuler< Operator > >

9.203 ImpliedTermStructure Class Reference

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

Inheritance diagram for ImpliedTermStructure:



9.203.1 Detailed Description

Implied term structure at a given date in the future.

The given date will be the implied reference date.

Note:

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

Public Member Functions

- **ImpliedTermStructure** (const **RelinkableHandle**< **TermStructure** > &, const **Date** &newTodaysDate, const **Date** &newReferenceDate)

TermStructure interface

- **DayCounter dayCounter** () const
the day counter used for date/time conversion
- **Date todaysDate** () const
today's date
- **Date referenceDate** () const
the reference date, i.e., the date at which discount = 1
- **Date maxDate** () const
the latest date for which the curve can return rates
- **Time maxTime** () const
the latest time for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- DiscountFactor **discountImpl** (Time, bool extrapolate=false) const
returns the discount factor as seen from the evaluation date

9.203.2 Member Function Documentation

9.203.2.1 void update () [virtual]

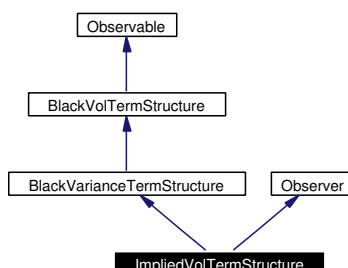
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.204 ImpliedVolTermStructure Class Reference

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

Inheritance diagram for ImpliedVolTermStructure:



9.204.1 Detailed Description

Implied vol term structure at a given date in the future.

The given date will be the implied reference date.

Note:

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

Warning:

It doesn't make financial sense to have an asset-dependant implied Vol Term Structure. This class should be used with term structures that are time dependant only

Public Member Functions

- **ImpliedVolTermStructure** (const **RelinkableHandle**< **BlackVolTermStructure** > &originalTS, const **Date** &newReferenceDate)

BlackVolTermStructure interface

- **Date** **referenceDate** () const
returns the reference date for which $t=0$
- **DayCounter** **dayCounter** () const
returns the day counter
- **Date** **maxDate** () const
the latest date for which the term structure can return vols

Observer interface

- void **update** ()

Visitability

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

Protected Member Functions

- virtual double **blackVarianceImpl** (Time t, double strike, bool extrapolate=false) const
implements the actual Black variance calculation in derived classes

9.204.2 Member Function Documentation

9.204.2.1 void update () [virtual]

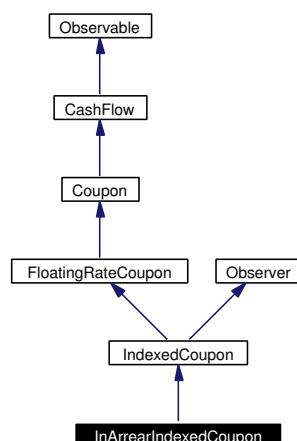
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.205 InArrearIndexedCoupon Class Reference

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

Inheritance diagram for InArrearIndexedCoupon:



9.205.1 Detailed Description

In-arrear indexed coupon class.

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **InArrearIndexedCoupon** (double nominal, const **Date** &paymentDate, const **Handle**<**Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, Spread spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())

FloatingRateCoupon interface

- **Date** fixingDate () const

Visitability

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

9.206 IncrementalStatistics Class Reference

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

9.206.1 Detailed Description

Statistics tool based on incremental accumulation.

It can accumulate a set of data and return statistics (e.g: mean, variance, skewness, kurtosis, error estimation, etc.)

Warning:

high moments are numerically unstable for high average/standardDeviation ratios

Public Member Functions

Inspectors

- Size **samples** () const
number of samples collected
- double **weightSum** () const
sum of data weights
- double **mean** () const
- double **variance** () const
- double **standardDeviation** () const
- double **downsideVariance** () const
- double **downsideDeviation** () const
- double **errorEstimate** () const
- double **skewness** () const
- double **kurtosis** () const
- double **min** () const
- double **max** () const

Modifiers

- void **add** (double value, double weight=1.0)
adds a datum to the set, possibly with a weight
- template<class DataIterator> void **addSequence** (DataIterator begin, DataIterator end)
adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator> void **addSequence** (DataIterator begin, DataIterator end, WeightIterator wbegin)
adds a sequence of data to the set, each with its weight
- void **reset** ()
resets the data to a null set

Protected Attributes

- Size `sampleNumber_`
- Size `downsideSampleNumber_`
- double `sampleWeight_`
- double `downsideSampleWeight_`
- double `sum_`
- double `quadraticSum_`
- double `downsideQuadraticSum_`
- double `cubicSum_`
- double `fourthPowerSum_`
- double `min_`
- double `max_`

9.206.2 Member Function Documentation

9.206.2.1 `double mean () const`

returns the mean, defined as

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

9.206.2.2 `double variance () const`

returns the variance, defined as

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

9.206.2.3 `double standardDeviation () const`

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

9.206.2.4 `double downsideVariance () const`

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^N \theta \times x_i^2}{\sum_{i=1}^N w_i}$$

, where $\theta = 0$ if $x > 0$ and $\theta = 1$ if $x < 0$

9.206.2.5 `double downsideDeviation () const`

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

9.206.2.6 double errorEstimate () const

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

9.206.2.7 double 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.

9.206.2.8 double 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.

9.206.2.9 double min () const

returns the minimum sample value

9.206.2.10 double max () const

returns the maximum sample value

9.206.2.11 void add (double *value*, double *weight* = 1.0)

adds a datum to the set, possibly with a weight

Precondition:

weight must be positive or null

9.206.2.12 void addSequence (DataIterator *begin*, DataIterator *end*, WeightIterator *wbegin*)

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

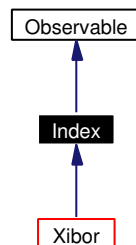
Precondition:

weights must be positive or null

9.207 Index Class Reference

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

Inheritance diagram for Index:



9.207.1 Detailed Description

purely virtual base class for indexes

Public Member Functions

- virtual `std::string name () const=0`
Returns the name of the index.
- virtual `Rate fixing (const Date &fixingDate) const=0`
returns the fixing at the given date

9.207.2 Member Function Documentation

9.207.2.1 `virtual std::string name () const [pure virtual]`

Returns the name of the index.

Warning:

This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implemented in **Xibor** (p. [632](#)).

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

returns the fixing at the given date

Note:

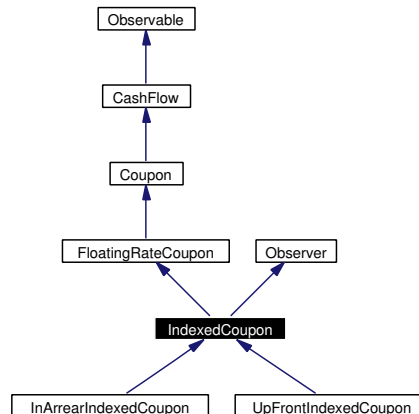
any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implemented in **Xibor** (p. [632](#)).

9.208 IndexedCoupon Class Reference

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

Inheritance diagram for IndexedCoupon:



9.208.1 Detailed Description

Base indexed coupon class.

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **IndexedCoupon** (double nominal, const **Date** &paymentDate, const **Handle**< **Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, Spread spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())

CashFlow interface

- double **amount** () const
returns the amount of the cash flow

Coupon interface

- **DayCounter** **dayCounter** () const
day counter for accrual calculation

FloatingRateCoupon interface

- Rate **fixing** () const

Inspectors

- `const Handle< Xibor > & index () const`

Observer interface

- `void update ()`

Visitability

- `virtual void accept (AcyclicVisitor &)`

9.208.2 Member Function Documentation

9.208.2.1 `double amount () const` [virtual]

returns the amount of the cash flow

Note:

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

Implements **CashFlow** (p. [200](#)).

9.208.2.2 `void update ()` [virtual]

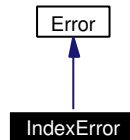
This method must be implemented in derived classes. An instance of **Observer** does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.209 IndexError Class Reference

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

Inheritance diagram for IndexError:



9.209.1 Detailed Description

Specialized error.

Thrown upon accessing an array or container outside its range.

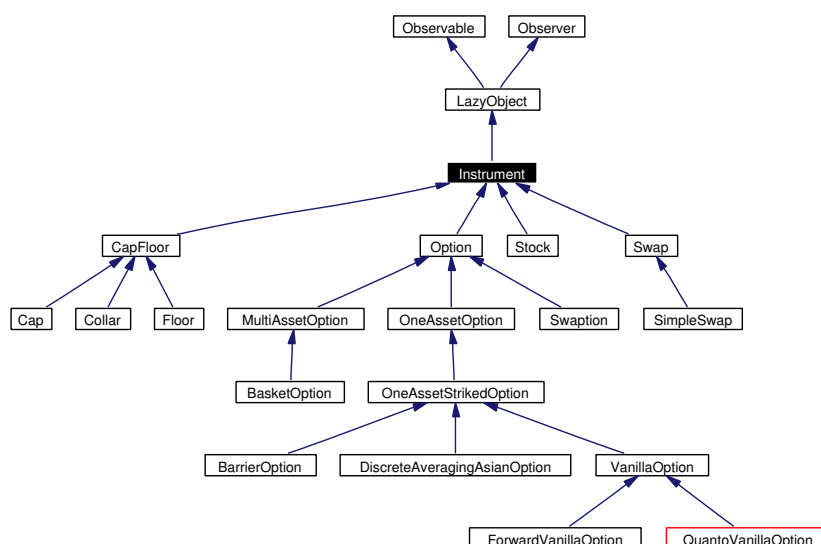
Public Member Functions

- `IndexError` (const std::string &what="")

9.210 Instrument Class Reference

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

Inheritance diagram for Instrument:



9.210.1 Detailed Description

Abstract instrument class.

This class is purely abstract and defines the interface of concrete instruments which will be derived from this one.

Public Member Functions

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

Inspectors

- double **NPV** () const
returns the net present value of the instrument.
- double **errorEstimate** () const
returns the error estimate on the NPV when available.
- virtual bool **isExpired** () const=0
returns whether the instrument is still tradable.

Modifiers

- void **setPricingEngine** (const **Handle**< **PricingEngine** > &)
set the pricing engine to be used.

Protected Member Functions

Calculations

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

Protected Attributes

- **Handle< PricingEngine > engine_**

Results

The value of this attribute and any other that derived classes might declare must be set during calculation.

- double **NPV_**
- double **errorEstimate_**

9.210.2 Member Function Documentation

9.210.2.1 void setPricingEngine (const Handle< PricingEngine > &)

set the pricing engine to be used.

Warning:

calling this method will have no effects in case the **performCalculation** method was overridden in a derived class.

9.210.2.2 void setupArguments (Arguments *) const [virtual]

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

Reimplemented in **DiscreteAveragingAsianOption** (p. 256), **BarrierOption** (p. 137), **BasketOption** (p. 140), **CapFloor** (p. 196), **ForwardVanillaOption** (p. 316), **MultiAssetOption** (p. 452), **OneAssetOption** (p. 478), **OneAssetStrikedOption** (p. 482), **QuantoForwardVanillaOption** (p. 525), **QuantoVanillaOption** (p. 531), **SimpleSwap** (p. 557), and **Swaption** (p. 585).

9.210.2.3 void calculate () const [protected, virtual]

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

Warning:

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

Should this method be redefined in derived classes, **LazyObject::calculate()**(p. 392) should be called in the overriding method.

Reimplemented from **LazyObject** (p. 392).

9.210.2.4 void setupExpired () const [protected, virtual]

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

Reimplemented in **MultiAssetOption** (p. 452), **OneAssetOption** (p. 478), **QuantoVanillaOption** (p. 531), and **Swap** (p. 582).

9.210.2.5 void performCalculations () const [protected, virtual]

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

Implements **LazyObject** (p. 392).

Reimplemented in **DiscreteAveragingAsianOption** (p. 256), **BarrierOption** (p. 137), **BasketOption** (p. 141), **ForwardVanillaOption** (p. 316), **MultiAssetOption** (p. 452), **OneAssetOption** (p. 478), **OneAssetStrikedOption** (p. 482), **QuantoVanillaOption** (p. 531), **Stock** (p. 574), **Swap** (p. 582), **Swaption** (p. 586), and **VanillaOption** (p. 624).

9.211 IntegerFormatter Class Reference

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

9.211.1 Detailed Description

Formats integers for output.

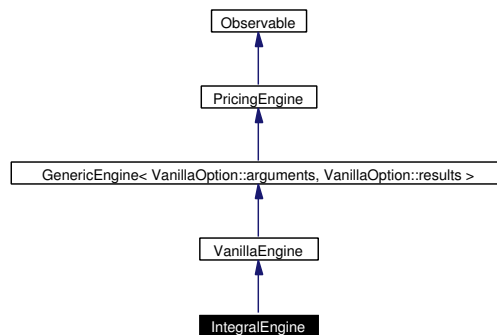
Static Public Member Functions

- `std::string toString` (long l, int digits=0)
- `std::string toOrdinal` (long l)
- `std::string toPowerOfTwo` (unsigned long l, int digits=0)

9.212 IntegralEngine Class Reference

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

Inheritance diagram for IntegralEngine:



9.212.1 Detailed Description

Pricing engine for European vanilla options using integral approach

Todo

define tolerance for calculate()

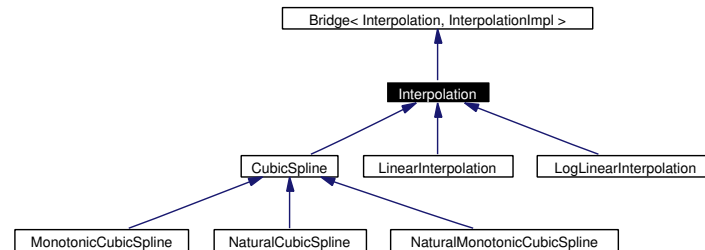
Public Member Functions

- void **calculate** () const

9.213 Interpolation Class Reference

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

Inheritance diagram for Interpolation:



9.213.1 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 double **argument_type**
- typedef double **result_type**

Public Member Functions

- double **operator()** (double x, bool allowExtrapolation=false) const
- double **primitive** (double x, bool allowExtrapolation=false) const
- double **derivative** (double x, bool allowExtrapolation=false) const
- double **secondDerivative** (double x, bool allowExtrapolation=false) const
- double **xMin** () const
- double **xMax** () const
- bool **isInRange** (double x) const

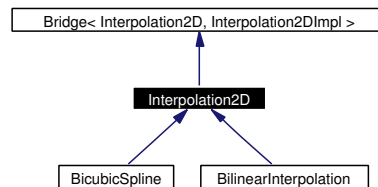
Protected Member Functions

- void **checkRange** (double x, bool allowExtrapolation) const

9.214 Interpolation2D Class Reference

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

Inheritance diagram for Interpolation2D:



9.214.1 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 double **first_argument_type**
- typedef double **second_argument_type**
- typedef double **result_type**

Public Member Functions

- double **operator()** (double x, double y, bool allowExtrapolation=false) const
- double **xMin** () const
- double **xMax** () const
- double **yMin** () const
- double **yMax** () const
- bool **isInRange** (double x, double y) const

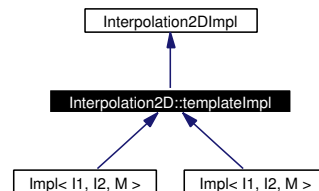
Protected Member Functions

- void **checkRange** (double x, double y, bool allowExtrapolation) const

9.215 Interpolation2D::templateImpl Class Template Reference

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

Inheritance diagram for Interpolation2D::templateImpl:



9.215.1 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)
- double **xMin** () const
- double **xMax** () const
- double **yMin** () const
- double **yMax** () const
- bool **isInRange** (double x, double y) const

Protected Member Functions

- Size **locateX** (double x) const
- Size **locateY** (double y) const

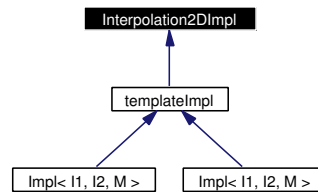
Protected Attributes

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

9.216 Interpolation2DImpl Class Reference

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

Inheritance diagram for Interpolation2DImpl:



9.216.1 Detailed Description

abstract base class for 2-D interpolation implementations

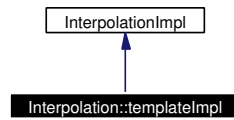
Public Member Functions

- virtual double **xMin** () const=0
- virtual double **xMax** () const=0
- virtual double **yMin** () const=0
- virtual double **yMax** () const=0
- virtual bool **isInRange** (double x, double y) const=0
- virtual double **value** (double x, double y) const=0

9.217 Interpolation::templateImpl Class Template Reference

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

Inheritance diagram for Interpolation::templateImpl:



9.217.1 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)
- double **xMin** () const
- double **xMax** () const
- bool **isInRange** (double x) const

Protected Member Functions

- Size **locate** (double x) const

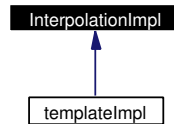
Protected Attributes

- I1 **xBegin_**
- I1 **xEnd_**
- I2 **yBegin_**

9.218 InterpolationImpl Class Reference

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

Inheritance diagram for InterpolationImpl:



9.218.1 Detailed Description

abstract base class for interpolation implementations

Public Member Functions

- virtual double **xMin** () const=0
- virtual double **xMax** () const=0
- virtual bool **isInRange** (double) const=0
- virtual double **value** (double) const=0
- virtual double **primitive** (double) const=0
- virtual double **derivative** (double) const=0
- virtual double **secondDerivative** (double) const=0

9.219 InverseCumulativeNormal Class Reference

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

9.219.1 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(p. 495), Statistics Division. URL: <http://home.online.no/~pjacklam/notes/invnorm/index.html>

This class can also be used to generate a gaussian normal distribution from a uniform distribution. This is especially useful when a gaussian normal distribution is generated from a low discrepancy uniform distribution: in this case the traditional Box-Muller approach and its variants would not preserve the sequence's low-discrepancy.

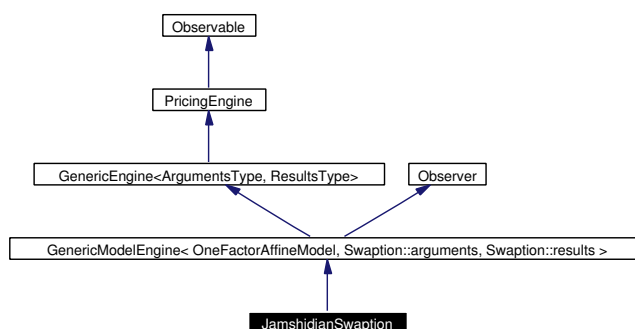
Public Member Functions

- **InverseCumulativeNormal** (double average=0.0, double sigma=1.0)
- double **operator()** (double x) const

9.220 JamshidianSwaption Class Reference

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

Inheritance diagram for JamshidianSwaption:



9.220.1 Detailed Description

Jamshidian swaption pricer.

Public Member Functions

- **JamshidianSwaption** (const **Handle**< **OneFactorAffineModel** > &mdl)
- **void calculate** () const

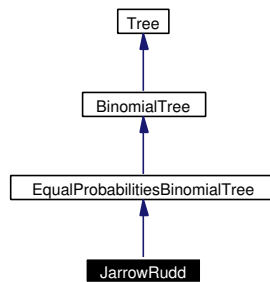
Friends

- class **rStarFinder**

9.221 JarrowRudd Class Reference

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

Inheritance diagram for JarrowRudd:



9.221.1 Detailed Description

Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

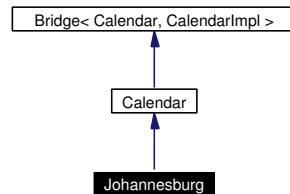
Public Member Functions

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

9.222 Johannesburg Class Reference

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

Inheritance diagram for Johannesburg:



9.222.1 Detailed Description

Johannesburg calendar

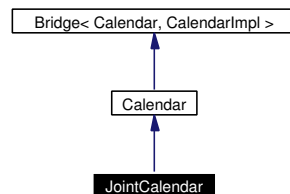
Holidays:

- 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)

9.223 JointCalendar Class Reference

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

Inheritance diagram for JointCalendar:



9.223.1 Detailed Description

Joint calendar.

Depending on the chosen rule, this calendar has a set of business days given by either the union or the intersection of the sets of business days of the given calendars.

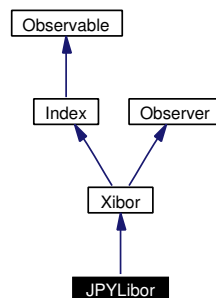
Public Member Functions

- **JointCalendar** (const **Calendar** &, const **Calendar** &, JointCalendarRule=JoinHolidays)
- **JointCalendar** (const **Calendar** &, const **Calendar** &, const **Calendar** &, JointCalendarRule=JoinHolidays)
- **JointCalendar** (const **Calendar** &, const **Calendar** &, const **Calendar** &, const **Calendar** &, JointCalendarRule=JoinHolidays)

9.224 JPYLibor Class Reference

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

Inheritance diagram for JPYLibor:



9.224.1 Detailed Description

JPY Libor index, also known as TIBOR

Todo

check settlement days

Public Member Functions

- **JPYLibor** (int n, TimeUnit units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**Actual360**())

9.225 KnuthUniformRng Class Reference

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

9.225.1 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**< double > **sample_type**

Public Member Functions

- **KnuthUniformRng** (long seed=0)
- **sample_type next** () const

9.225.2 Constructor & Destructor Documentation

9.225.2.1 **KnuthUniformRng** (long *seed* = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

9.225.3 Member Function Documentation

9.225.3.1 **KnuthUniformRng::sample_type next** () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

9.226 KronrodIntegral Class Reference

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

9.226.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

References: Gauss-Kronrod Integration <http://mathcssun1.emporia.edu/~oneilcat/Experiment-Applet3/ExperimentApplet3.html> NMS - Numerical Analysis Library http://www.math.iastate.edu/burkardt/f_src/nms/nms.html

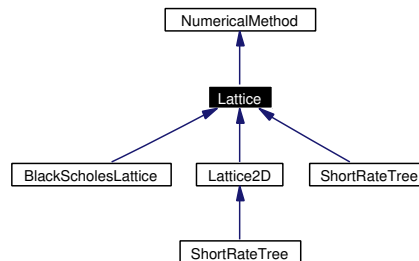
Public Member Functions

- **KronrodIntegral** (double tolerance, Size maxFunctionEvaluations=**Null**< int >())
- template<class F> double **operator()** (const F &f, double a, double b) const
- Size **functionEvaluations** ()

9.227 Lattice Class Reference

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

Inheritance diagram for Lattice:



9.227.1 Detailed Description

Lattice-method base class.

This class defines a lattice method that is able to rollback (with discount) a discretized asset object. It will usually be based on one or more trees.

Public Member Functions

- **Lattice** (const **TimeGrid** &timeGrid, Size n)
- double **presentValue** (const **Handle**< **DiscretizedAsset** > &asset)
Computes the present value of an asset using Arrow-Debreu prices.
- void **initialize** (const **Handle**< **DiscretizedAsset** > &asset, Time t) const
*Initialize a **DiscretizedAsset**(p. 260) object.*
- void **rollback** (const **Handle**< **DiscretizedAsset** > &asset, Time to) const
- void **rollAlmostBack** (const **Handle**< **DiscretizedAsset** > &asset, Time to) const
- virtual Size **size** (Size i) const=0
- virtual DiscountFactor **discount** (Size i, Size index) const=0
Discount factor at time t_i and node indexed by index.
- const **Array** & **statePrices** (Size i)
- virtual Size **descendant** (Size i, Size index, Size branch) const=0
***Tree**(p. 607) properties.*
- virtual double **probability** (Size i, Size index, Size branch) const=0

Protected Member Functions

- void **computeStatePrices** (Size until)
- virtual void **stepback** (Size i, const **Array** &values, **Array** &newValues) const

Protected Attributes

- `std::vector< Array > statePrices_`

9.227.2 Member Function Documentation

9.227.2.1 `void rollback (const Handle< DiscretizedAsset > & asset, Time to) const`
[virtual]

Roll back a **DiscretizedAsset**(p. 260) object until a certain time, performing any needed adjustment
Implements **NumericalMethod** (p. 470).

9.227.2.2 `void rollAlmostBack (const Handle< DiscretizedAsset > & asset, Time to) const`
[virtual]

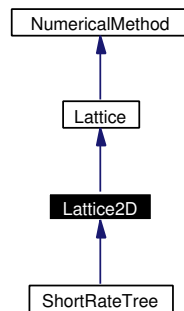
Roll-back a **DiscretizedAsset**(p. 260) object until a certain time, but do not perform the final adjustment.

Implements **NumericalMethod** (p. 470).

9.228 Lattice2D Class Reference

```
#include <ql/Lattices/lattice2d.hpp>
```

Inheritance diagram for Lattice2D:



9.228.1 Detailed Description

Two-dimensional lattice.

This lattice is based on two trinomial trees and primarily used for the **G2**([p. 321](#)) short-rate model.

Public Member Functions

- **Lattice2D** (const **Handle**< **TrinomialTree** > &tree1, const **Handle**< **TrinomialTree** > &tree2, double correlation)
- **Size** **size** (Size i) const

Protected Member Functions

- **Size** **descendant** (Size i, Size index, Size branch) const
Tree([p. 607](#)) *properties*.
- double **probability** (Size i, Size index, Size branch) const

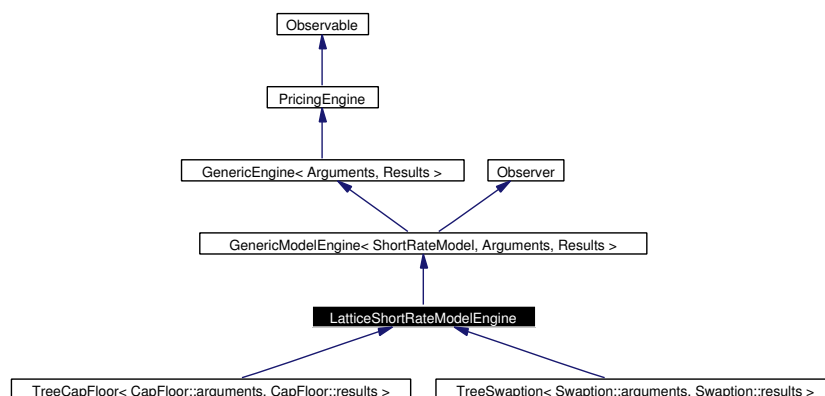
Protected Attributes

- **Handle**< **Tree** > tree1_
- **Handle**< **Tree** > tree2_

9.229 LatticeShortRateModelEngine Class Template Reference

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

Inheritance diagram for LatticeShortRateModelEngine:



9.229.1 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 **Handle**< **ShortRateModel** > &model, Size timeSteps)
- **LatticeShortRateModelEngine** (const **Handle**< **ShortRateModel** > &model, const **TimeGrid** &timeGrid)
- void **update** ()

Protected Attributes

- **TimeGrid** timeGrid_
- Size timeSteps_
- **Handle**< **Lattice** > lattice_

9.229.2 Member Function Documentation

9.229.2.1 void update () [virtual]

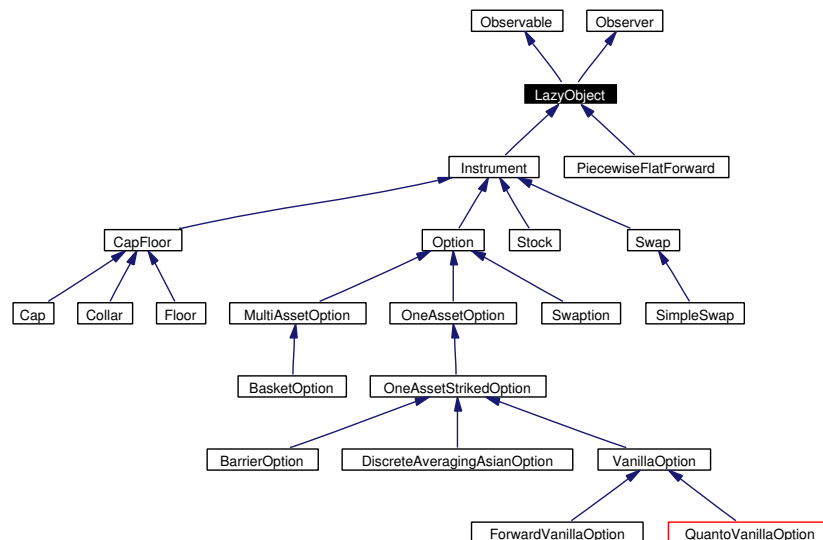
This method must be implemented in derived classes. An instance of **Observer** does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from **GenericModelEngine< ShortRateModel, Arguments, Results >** (p. [333](#)).

9.230 LazyObject Class Reference

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

Inheritance diagram for LazyObject:



9.230.1 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_**

9.230.2 Member Function Documentation

9.230.2.1 void update () [virtual]

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

Implements **Observer** (p. 475).

9.230.2.2 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.

9.230.2.3 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.

9.230.2.4 void unfreeze ()

This method reverts the effect of the **freeze** method, thus re-enabling recalculations.

9.230.2.5 void calculate () const [protected, virtual]

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

Warning:

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

Should this method be redefined in derived classes, **LazyObject::calculate()**(p. 392) should be called in the overriding method.

Reimplemented in **Instrument** (p. 368).

9.230.2.6 virtual void performCalculations () const [protected, pure virtual]

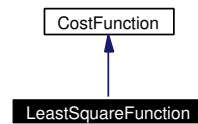
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implemented in **Instrument** (p. 369), **DiscreteAveragingAsianOption** (p. 256), **BarrierOption** (p. 137), **BasketOption** (p. 141), **ForwardVanillaOption** (p. 316), **MultiAssetOption** (p. 452), **OneAssetOption** (p. 478), **OneAssetStrikedOption** (p. 482), **QuantoVanillaOption** (p. 531), **Stock** (p. 574), **Swap** (p. 582), **Swaption** (p. 586), and **VanillaOption** (p. 624).

9.231 LeastSquareFunction Class Reference

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

Inheritance diagram for LeastSquareFunction:



9.231.1 Detailed Description

Cost function for least-square problems.

Implements a cost function using the interface provided by the [LeastSquareProblem](#)(p. 395) class. [Array](#)(p. 126) vector class requires function `DotProduct()` that computes dot product and - operator. M matrix class requires function `transpose()` that computes transpose and * operator with vector class.

Public Member Functions

- **LeastSquareFunction** ([LeastSquareProblem](#) &lsp)
Default constructor.
- virtual **~LeastSquareFunction** ()
Destructor.
- virtual double **value** (const [Array](#) &x) const
compute value of the least square function
- virtual void **gradient** ([Array](#) &grad_f, const [Array](#) &x) const
compute vector of derivatives of the least square function
- virtual double **valueAndGradient** ([Array](#) &grad_f, const [Array](#) &x) const
compute value and gradient of the least square function

Protected Attributes

- **LeastSquareProblem** & lsp_
least square problem

9.232 LeastSquareProblem Class Reference

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

9.232.1 Detailed Description

Base class for least square problem.

Public Member Functions

- virtual int **size** ()=0
size of the problem ie size of target vector
- virtual void **targetAndValue** (const **Array** &x, **Array** &target, **Array** &fct2fit)=0
compute the target vector and the values of the fonction to fit
- virtual void **targetValueAndGradient** (const **Array** &x, **Matrix** &grad_fct2fit, **Array** &target, **Array** &fct2fit)=0

9.232.2 Member Function Documentation

9.232.2.1 virtual void **targetValueAndGradient** (const **Array** & *x*, **Matrix** & *grad_fct2fit*, **Array** & *target*, **Array** & *fct2fit*) [pure virtual]

compute the target vector, the values of the fonction to fit and the matrix of derivatives

9.233 LecuyerUniformRng Class Reference

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

9.233.1 Detailed Description

Uniform random number generator.

Random number generator of L'Ecuyer with added Bays-Durham shuffle (known as ran2 in Numerical recipes)

For more details see Section 7.1 of Numerical Recipes in C, 2nd Edition, Cambridge University Press (available at <http://www.nr.com/>)

Public Types

- typedef `Sample< double > sample_type`

Public Member Functions

- `LecuyerUniformRng` (long seed=0)
- `sample_type next () const`

9.233.2 Constructor & Destructor Documentation

9.233.2.1 `LecuyerUniformRng (long seed = 0) [explicit]`

if the given seed is 0, a random seed will be chosen based on `clock()`

9.233.3 Member Function Documentation

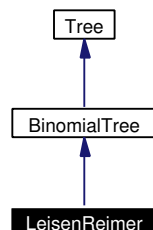
9.233.3.1 `LecuyerUniformRng::sample_type next () const`

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

9.234 LeisenReimer Class Reference

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

Inheritance diagram for LeisenReimer:



9.234.1 Detailed Description

Leisen & Reimer tree: multiplicative approach.

Public Member Functions

- **LeisenReimer** (const **Handle**< **DiffusionProcess** > &process, Time end, Size steps, double strike)
- double **underlying** (Size i, Size index) const
- double **probability** (Size, Size, Size) const

Protected Attributes

- double **up**_
- double **down**_
- double **pu**_
- double **pd**_

9.235 LexicographicalView Class Template Reference

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

9.235.1 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 **stepping_iterator**< RandomAccessIterator > **y_iterator**
iterates over v_{ij} with i fixed.

Public Member Functions

- **LexicographicalView** (const RandomAccessIterator &begin, const RandomAccessIterator &end, int xSize)
attaches the view with the given dimension to a sequence
- typedef **QL_REVERSE_ITERATOR** (RandomAccessIterator, typename 1< RandomAccessIterator >::value_type) reverse_x_iterator
iterates backwards over v_{ij} with j fixed.
- typedef **QL_REVERSE_ITERATOR** (y_iterator, typename 1< RandomAccessIterator >::value_type) reverse_y_iterator
iterates backwards over v_{ij} with i fixed.

Element access

- **y_iterator operator[]** (int i)

Iterator access

- **x_iterator xbegin** (int j)
- **x_iterator xend** (int j)
- **reverse_x_iterator rxbegin** (int j)
- **reverse_x_iterator rxend** (int j)
- **y_iterator ybegin** (int i)
- **y_iterator yend** (int i)
- **reverse_y_iterator rybegin** (int i)

- reverse_y_iterator **ryend** (int i)

Inspectors

- int **xSize** () const
dimension of the array along x
- int **ySize** () const
dimension of the array along y

9.236 Linear Class Reference

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

9.236.1 Detailed Description

Linear interpolation traits

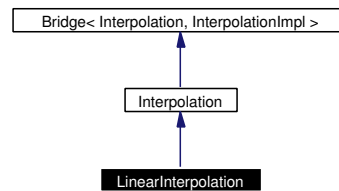
Static Public Member Functions

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

9.237 LinearInterpolation Class Reference

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

Inheritance diagram for LinearInterpolation:



9.237.1 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)`

9.237.2 Constructor & Destructor Documentation

9.237.2.1 LinearInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

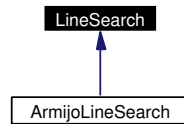
Precondition:

the *x* values must be sorted.

9.238 LineSearch Class Reference

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

Inheritance diagram for LineSearch:



9.238.1 Detailed Description

Base class for line search.

Public Member Functions

- **LineSearch** (double eps=1e-8)
Default constructor.
- virtual **~LineSearch** ()
Destructor.
- const **Array** & **lastX** ()
return last x value
- double **lastFunctionValue** ()
return last cost function value
- const **Array** & **lastGradient** ()
return last gradient
- double **lastGradientNorm2** ()
return square norm of last gradient
- bool **succeed** ()
- virtual double **operator()** (const **Problem** &P, double t_ini)=0
Perform line search.
- double **update** (**Array** ¶ms, const **Array** &direction, double beta, const **Constraint** &constraint)

Protected Attributes

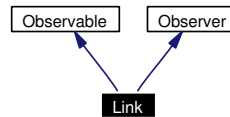
- **Array** **xtd_**
new x and its gradient

- **Array gradient_**
new x and its gradient
- **double qt_**
cost function value and gradient norm corresponding to $xtd_$
- **double qpt_**
cost function value and gradient norm corresponding to $xtd_$
- **bool succeed_**
flag to know if linesearch succeed

9.239 Link Class Template Reference

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

Inheritance diagram for Link:



9.239.1 Detailed Description

```
template<class Type> class QuantLib::Link< Type >
```

Relinkable access to a Handle.

Precondition:

Class "Type" must inherit from **Observable**(p. 471)

Public Member Functions

- **Link** (const **Handle**< Type > &h=**Handle**< Type >(), bool registerAsObserver=true)
- void **linkTo** (const **Handle**< Type > &h, bool registerAsObserver=true)
- bool **isNull** () const
Checks if the contained handle points to anything.
- const **Handle**< Type > & **currentLink** () const
Returns the contained handle.
- void **update** ()
Observer(p. 473) interface.

9.239.2 Constructor & Destructor Documentation

9.239.2.1 **Link** (const **Handle**< Type > &h = **Handle**< Type >(), bool registerAsObserver = true) [explicit]

Warning:

see the documentation of the linkTo method for issues relatives to registerAsObserver.

9.239.3 Member Function Documentation

9.239.3.1 void **linkTo** (const **Handle**< Type > &h, bool registerAsObserver = true)

Warning:

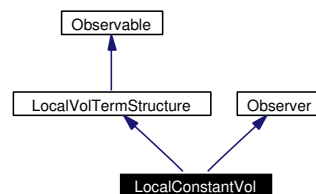
registerAsObserver is left as a backdoor in case the programmer cannot guarantee that the object pointed to will remain alive for the whole lifetime of the handle—namely, it should be

set to `false` when the passed handle was created with `owns = false` (the latter should only happen in a controlled environment, so that the programmer is aware of it). Failure to do so can very likely result in a program crash. If the programmer does want the relinkable handle to register as observer of such a handle, it is his responsibility to ensure that the relinkable handle gets destroyed before the pointed object does.

9.240 LocalConstantVol Class Reference

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

Inheritance diagram for LocalConstantVol:



9.240.1 Detailed Description

Constant local volatility, no time-strike dependence.

This class implements the LocalVolatilityTermStructure interface for a constant local volatility (no time/asset dependence). Local volatility and Black volatility are the same when volatility is at most time dependent, so this class is basically a proxy for **BlackVolatilityTermStructure**(p. 169).

Public Member Functions

- **LocalConstantVol** (const **Date** &referenceDate, double volatility, const **DayCounter** &dayCounter=**Actual365**())
- **LocalConstantVol** (const **Date** &referenceDate, const **RelinkableHandle**< **Quote** > &volatility, const **DayCounter** &dayCounter=**Actual365**())

LocalVolTermStructure interface

- **Date** **referenceDate** () const
returns the reference date for which $t=0$
- **DayCounter** **dayCounter** () const
returns the day counter
- **Date** **maxDate** () const
the latest date for which the term structure can return vols

Observer interface

- void **update** ()

Visitability

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

9.240.2 Member Function Documentation

9.240.2.1 void update () [virtual]

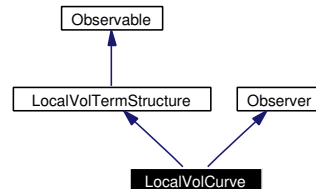
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.241 LocalVolCurve Class Reference

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

Inheritance diagram for LocalVolCurve:



9.241.1 Detailed Description

Local volatility curve derived from a Black curve.

Public Member Functions

- **LocalVolCurve** (const **RelinkableHandle**< **BlackVarianceCurve** > &curve)

LocalVolTermStructure interface

- **Date** **referenceDate** () const
returns the reference date for which $t=0$
- **DayCounter** **dayCounter** () const
returns the day counter
- **Date** **maxDate** () const
the latest date for which the term structure can return vols

Observer interface

- void **update** ()

Visitability

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

Protected Member Functions

- double **localVolImpl** (Time, double, bool extrapolate) const

9.241.2 Member Function Documentation

9.241.2.1 void update () [virtual]

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

Implements **Observer** (p. [475](#)).

9.241.2.2 double localVolImpl (Time *t*, double *dummy*, bool *extrapolate*) const [protected, virtual]

The relation

$$\int_0^T \sigma_L^2(t) dt = \sigma_B^2 T$$

holds, where $\sigma_L(t)$ is the local volatility at time t and $\sigma_B(T)$ is the Black volatility for maturity T . From the above, the formula

$$\sigma_L(t) = \sqrt{\frac{d}{dt} \sigma_B^2(t) t}$$

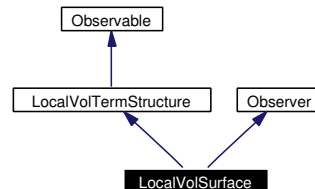
can be deduced which is here implemented.

Implements **LocalVolTermStructure** (p. [412](#)).

9.242 LocalVolSurface Class Reference

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

Inheritance diagram for LocalVolSurface:



9.242.1 Detailed Description

Local volatility surface derived from a Black vol surface.

For details about this implementation refers to "Stochastic Volatility and Local Volatility," in "Case Studies in 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 **RelinkableHandle**< **BlackVolTermStructure** > &blackTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > ÷ndTS, const **RelinkableHandle**< **Quote** > &underlying)
- **LocalVolSurface** (const **RelinkableHandle**< **BlackVolTermStructure** > &blackTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > ÷ndTS, double underlying)

LocalVolTermStructure interface

- **Date** **referenceDate** () const
returns the reference date for which $t=0$
- **DayCounter** **dayCounter** () const
returns the day counter
- **Date** **maxDate** () const
the latest date for which the term structure can return vols

Observer interface

- void **update** ()

Visitability

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

Protected Member Functions

- double **localVolImpl** (Time, double, bool extrapolate) const
implements the actual local vol calculation in derived classes

9.242.2 Member Function Documentation

9.242.2.1 void update () [virtual]

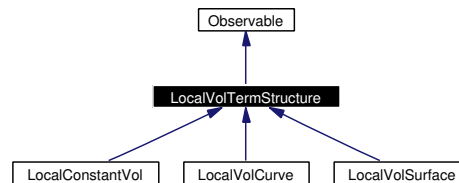
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.243 LocalVolTermStructure Class Reference

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

Inheritance diagram for LocalVolTermStructure:



9.243.1 Detailed Description

Local-volatility term structure.

This abstract class defines the interface of concrete local-volatility term structures which will be derived from this one.

Volatilities are assumed to be expressed on an annual basis.

Public Member Functions

Local Volatility

- double **localVol** (const **Date** &d, double underlyingLevel, bool extrapolate=false) const
- double **localVol** (Time t, double underlyingLevel, bool extrapolate=false) const

Dates

- virtual **Date** **referenceDate** () const=0
returns the reference date for which t=0
- virtual **DayCounter** **dayCounter** () const=0
returns the day counter
- virtual **Date** **maxDate** () const=0
the latest date for which the term structure can return vols
- Time **maxTime** () const
the latest time for which the term structure can return vols

Visitability

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

Protected Member Functions

- virtual double **localVolImpl** (Time t, double strike, bool extrapolate=false) const=0
implements the actual local vol calculation in derived classes

9.244 LogLinear Class Reference

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

9.244.1 Detailed Description

Log-linear interpolation traits.

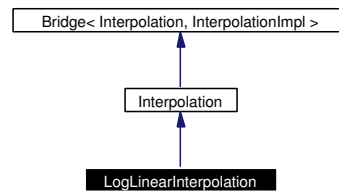
Static Public Member Functions

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

9.245 LogLinearInterpolation Class Reference

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

Inheritance diagram for LogLinearInterpolation:



9.245.1 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)`

9.245.2 Constructor & Destructor Documentation

9.245.2.1 LogLinearInterpolation (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

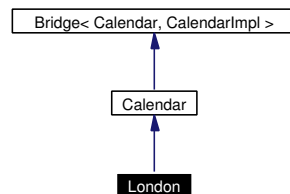
Precondition:

the *x* values must be sorted.

9.246 London Class Reference

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

Inheritance diagram for London:



9.246.1 Detailed Description

London calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Early May Bank Holiday, first Monday of May
- Spring Bank Holiday, last Monday of May
- Summer Bank Holiday, last Monday of August
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

See <http://www.dti.gov.uk/er/bankhol.htm>

9.247 `lowest_category_iterator` Struct Template Reference

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

9.247.1 Detailed Description

template<class Category1, class Category2> struct QuantLib::lowest_category_iterator< Category1, Category2 >

most generic of two given iterator categories

Specializations of this struct define a typedef `iterator_category` which corresponds to the most generic of the two input categories, e.g., **lowest_category_iterator**(p. 416)<std::random_access_iterator_tag, std::forward_iterator_tag>::iterator_category corresponds to std::forward_iterator_tag.

9.248 MakeSchedule Class Reference

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

9.248.1 Detailed Description

helper class

This class provides a more comfortable interface to the argument list of Schedule's constructor.

Public Member Functions

- **MakeSchedule** (const **Calendar** &calendar, const **Date** &startDate, const **Date** &endDate, int frequency, RollingConvention rollingConvention, bool isAdjusted)
- **MakeSchedule & withStubDate** (const **Date** &d)
- **MakeSchedule & backwards** (bool flag=true)
- **MakeSchedule & forwards** (bool flag=true)
- **MakeSchedule & longFinalPeriod** (bool flag=true)
- **MakeSchedule & shortFinalPeriod** (bool flag=true)
- **operator Schedule** ()

9.249 Matrix Class Reference

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

9.249.1 Detailed Description

Matrix used in linear algebra.

This class implements the concept of **Matrix**(p. 418) as used in linear algebra. As such, it is **not** meant to be used as a container.

Public Types

- typedef double * **iterator**
- typedef const double * **const_iterator**
- typedef double * **row_iterator**
- typedef const double * **const_row_iterator**
- typedef **stepping_iterator**< double * > **column_iterator**
- typedef **stepping_iterator**< const double * > **const_column_iterator**

Public Member Functions

- typedef **QL_REVERSE_ITERATOR** (iterator, double) **reverse_iterator**
- typedef **QL_REVERSE_ITERATOR** (const_iterator, double) **const_reverse_iterator**
- typedef **QL_REVERSE_ITERATOR** (row_iterator, double) **reverse_row_iterator**
- typedef **QL_REVERSE_ITERATOR** (const_row_iterator, double) **const_reverse_row_iterator**
- typedef **QL_REVERSE_ITERATOR** (column_iterator, double) **reverse_column_iterator**
- typedef **QL_REVERSE_ITERATOR** (const_column_iterator, double) **const_reverse_column_iterator**

Constructors, destructor, and assignment

- **Matrix** ()
creates a null matrix
- **Matrix** (Size rows, Size columns)
creates a matrix with the given dimensions
- **Matrix** (Size rows, Size columns, double value)
creates the matrix and fills it with value
- **Matrix** (const **Matrix** &)
- **Matrix** (const **Disposable**< **Matrix** > &)
- **Matrix** & **operator=** (const **Matrix** &)
- **Matrix** & **operator=** (const **Disposable**< **Matrix** > &)

Algebraic operators

- const **Matrix** & **operator+=** (const **Matrix** &)
- const **Matrix** & **operator-=** (const **Matrix** &)

- const **Matrix** & **operator** *= (double)
- const **Matrix** & **operator** /= (double)

Iterator access

- const_iterator **begin** () const
- iterator **begin** ()
- const_iterator **end** () const
- iterator **end** ()
- const_reverse_iterator **rbegin** () const
- reverse_iterator **rbegin** ()
- const_reverse_iterator **rend** () const
- reverse_iterator **rend** ()
- const_row_iterator **row_begin** (Size i) const
- row_iterator **row_begin** (Size i)
- const_row_iterator **row_end** (Size i) const
- row_iterator **row_end** (Size i)
- const_reverse_row_iterator **row_rbegin** (Size i) const
- reverse_row_iterator **row_rbegin** (Size i)
- const_reverse_row_iterator **row_rend** (Size i) const
- reverse_row_iterator **row_rend** (Size i)
- const_column_iterator **column_begin** (Size i) const
- column_iterator **column_begin** (Size i)
- const_column_iterator **column_end** (Size i) const
- column_iterator **column_end** (Size i)
- const_reverse_column_iterator **column_rbegin** (Size i) const
- reverse_column_iterator **column_rbegin** (Size i)
- const_reverse_column_iterator **column_rend** (Size i) const
- reverse_column_iterator **column_rend** (Size i)

Element access

- const_row_iterator **operator**[] (Size) const
- row_iterator **operator**[] (Size)
- **Disposable**< **Array** > **diagonal** (void) const

Inspectors

- Size **rows** () const
- Size **columns** () const

Utilities

- void **swap** (**Matrix** &)

Related Functions

(Note that these are not member functions.)

- std::ostream & **operator**<< (std::ostream &, const **Matrix** &)
- 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** &, double)

- `const Disposable< Matrix > operator * (double, const Matrix &)`
- `const Disposable< Matrix > operator/ (const Matrix &, double)`
- `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)`
- `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, double componentRetainedPercentag, SalvagingAlgorithm::Type)`
- `const Disposable< Matrix > matrixSqrt (const Matrix &)`

9.249.2 Member Function Documentation

9.249.2.1 `const Matrix & operator+= (const Matrix &)`

Precondition:

all matrices involved in an algebraic expression must have the same size.

9.249.3 Friends And Related Function Documentation

9.249.3.1 `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

- a) 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" b) implement Higham algorithm: Higham "Computing the nearest correlation matrix"

9.249.3.2 `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, double componentRetainedPercentag, SalvagingAlgorithm::Type)` [related]

Precondition:

the given matrix must be symmetric.

9.249.3.3 `const Disposable< Matrix > matrixSqrt (const Matrix &)` [related]

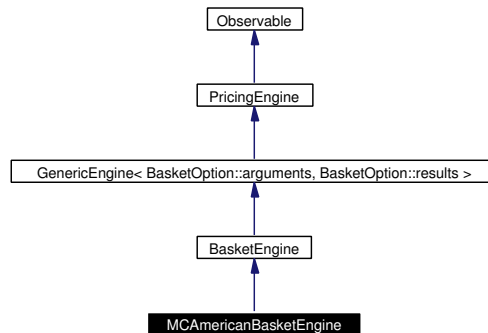
Deprecated

use CholeskyDecomposition or pseudoSqrt instead

9.250 MCAmericanBasketEngine Class Reference

```
#include <ql/PricingEngines/Basket/mcamericanbasketengine.hpp>
```

Inheritance diagram for MCAmericanBasketEngine:



9.250.1 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.

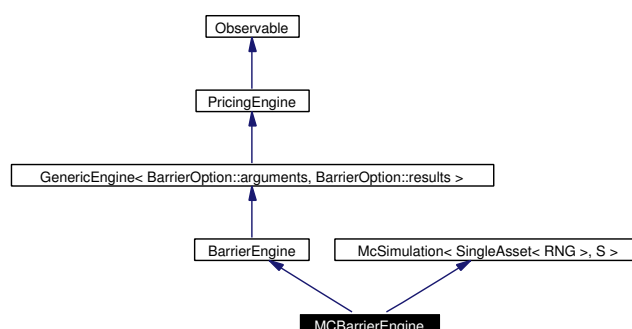
Public Member Functions

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

9.251 MCBarrierEngine Class Template Reference

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

Inheritance diagram for MCBarrierEngine:



9.251.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCBarrierEngine< RNG, S >
```

Pricing engine for barrier options using Monte Carlo.

Uses the Brownian-bridge correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic Option* (p. 490) 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

Public Member Functions

- **MCBarrierEngine** (Size maxTimeStepsPerYear, bool antitheticVariate=false, bool controlVariate=false, Size requiredSamples=Null< int >(), double requiredTolerance=Null< double >(), Size maxSamples=Null< int >(), bool isBiased=false, long seed=0)
- void **calculate** () const

Protected Types

- typedef **McSimulation**< SingleAsset< RNG >, S >::path_generator_type **path_generator_type**
- typedef **McSimulation**< SingleAsset< RNG >, S >::path_pricer_type **path_pricer_type**
- typedef **McSimulation**< SingleAsset< RNG >, S >::stats_type **stats_type**

Protected Member Functions

- **Handle**< path_generator_type > **pathGenerator** () const
- **TimeGrid** **timeGrid** () const
- **Handle**< path_pricer_type > **pathPricer** () const

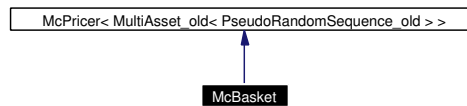
Protected Attributes

- Size `maxTimeStepsPerYear_`
- Size `requiredSamples_`
- Size `maxSamples_`
- double `requiredTolerance_`
- bool `isBiased_`
- long `seed_`

9.252 McBasket Class Reference

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

Inheritance diagram for McBasket:



9.252.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

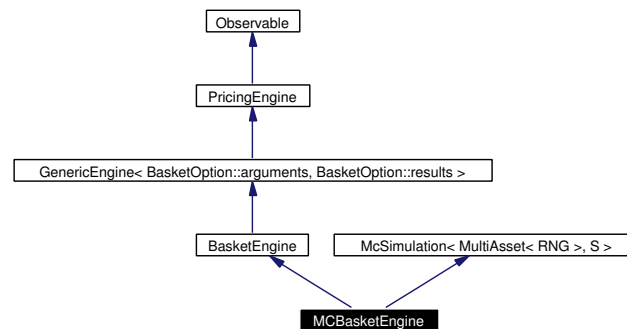
Public Member Functions

- **McBasket** (Option::Type type, const std::vector< double > &underlying, double strike, const **Array** ÷ndYield, const **Matrix** &covariance, Rate riskFreeRate, double residualTime, bool antitheticVariance, long seed=0)

9.253 MCBasketEngine Class Template Reference

```
#include <ql/PricingEngines/Basket/mcbasketengine.hpp>
```

Inheritance diagram for MCBasketEngine:



9.253.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCBasketEngine< RNG, S >
```

MC Pricing engine for European Baskets.

Public Types

- typedef **McSimulation**< MultiAsset< RNG >, S >::path_generator_type **path_generator_type**
- typedef **McSimulation**< MultiAsset< RNG >, S >::path_pricer_type **path_pricer_type**
- typedef **McSimulation**< MultiAsset< RNG >, S >::stats_type **stats_type**

Public Member Functions

- **MCBasketEngine** (Size maxTimeStepsPerYear, bool antitheticVariate=false, bool controlVariate=false, Size requiredSamples=**Null**< int >(), double requiredTolerance=**Null**< double >(), Size maxSamples=**Null**< int >(), bool brownianBridge=false, long seed=0)
- void **calculate** () const

Protected Member Functions

- **Handle**< path_generator_type > **pathGenerator** () const
- **TimeGrid** **timeGrid** () const
- **Handle**< path_pricer_type > **pathPricer** () const

Protected Attributes

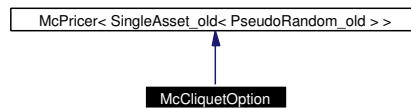
- Size **maxTimeStepsPerYear_**

- Size `requiredSamples_`
- Size `maxSamples_`
- double `requiredTolerance_`
- bool `brownianBridge_`
- long `seed_`

9.254 McCliquetOption Class Reference

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

Inheritance diagram for McCliquetOption:



9.254.1 Detailed Description

simple example of Monte Carlo pricer

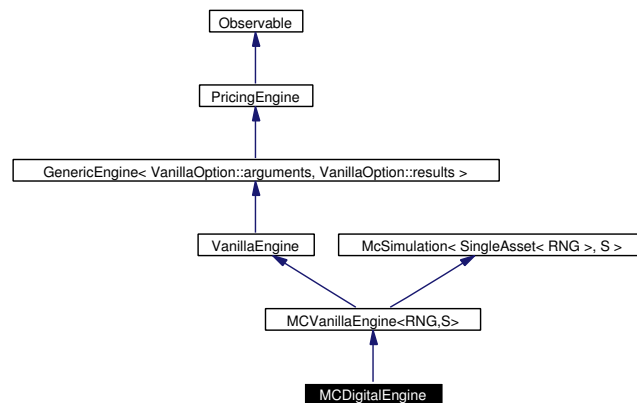
Public Member Functions

- **McCliquetOption** (Option::Type type, double underlying, double moneyness, const std::vector< Spread > ÷ndYield, const std::vector< Rate > &riskFreeRate, const std::vector< Time > ×, const std::vector< double > &volatility, double accruedCoupon, double lastFixing, double localCap, double localFloor, double globalCap, double globalFloor, bool redemptionOnly, bool antitheticVariance, long seed=0)

9.255 MCDigitalEngine Class Template Reference

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

Inheritance diagram for MCDigitalEngine:



9.255.1 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**(p. 179) correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic Option*(p. 490) 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

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 maxTimeStepsPerYear, bool antitheticVariate=false, bool control-Variate=false, Size requiredSamples=**Null**< int >(), double requiredTolerance=**Null**< double >(), Size maxSamples=**Null**< int >(), long seed=0)
- void **calculate** () const

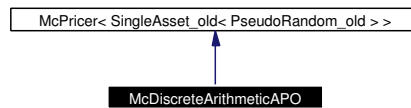
Protected Member Functions

- **TimeGrid** timeGrid () const
- **Handle**< path_pricer_type > **pathPricer** () const

9.256 McDiscreteArithmeticAPO Class Reference

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

Inheritance diagram for McDiscreteArithmeticAPO:



9.256.1 Detailed Description

example of Monte Carlo pricer using a control variate

Todo

Continous-averaging version

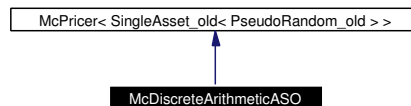
Public Member Functions

- **McDiscreteArithmeticAPO** (Option::Type type, double underlying, double strike, Spread dividendYield, Rate riskFreeRate, const std::vector< Time > ×, double volatility, bool antitheticVariance, bool controlVariate, long seed=0)

9.257 McDiscreteArithmeticASO Class Reference

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

Inheritance diagram for McDiscreteArithmeticASO:



9.257.1 Detailed Description

example of Monte Carlo pricer using a control variate.

Todo

Continous Averaging version

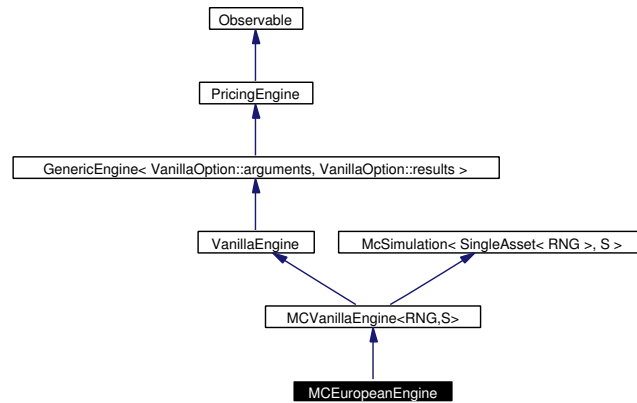
Public Member Functions

- **McDiscreteArithmeticASO** (Option::Type type, double underlying, Spread dividendYield, Rate riskFreeRate, const std::vector< Time > ×, double volatility, bool antithetic-Variance, bool controlVariate, long seed=0)

9.258 MCEuropeanEngine Class Template Reference

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

Inheritance diagram for MCEuropeanEngine:



9.258.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCEuropeanEngine< RNG, S >
```

European option pricing engine using Monte Carlo simulation.

Public Types

- typedef MCVanillaEngine< RNG, S >::path_generator_type **path_generator_type**
- typedef MCVanillaEngine< RNG, S >::path_pricer_type **path_pricer_type**
- typedef MCVanillaEngine< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCEuropeanEngine** (Size maxTimeStepPerYear, bool antitheticVariate=false, bool controlVariate=false, Size requiredSamples=**Null**< int >(), double requiredTolerance=**Null**< double >(), Size maxSamples=**Null**< int >(), long seed=0)

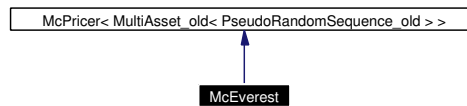
Protected Member Functions

- **TimeGrid** timeGrid () const
- **Handle**< path_pricer_type > **pathPricer** () const

9.259 McEverest Class Reference

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

Inheritance diagram for McEverest:



9.259.1 Detailed Description

Everest-type option pricer.

The payoff of an Everest option is simply given by the final price / initial price ratio of the worst performer

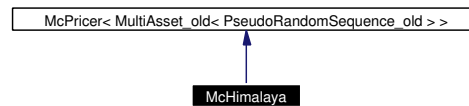
Public Member Functions

- **McEverest** (const **Array** ÷ndYield, const **Matrix** &covariance, Rate riskFreeRate, Time residualTime, bool antitheticVariance, long seed=0)

9.260 McHimalaya Class Reference

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

Inheritance diagram for McHimalaya:



9.260.1 Detailed Description

Himalayan-type option pricer.

The payoff of a Himalaya option is computed in the following way: Given a basket of N assets, and N time periods, at end of each period the option who performed the best is added to the average and then discarded from the basket. At the end of the N periods the option pays the max between the strike and the average of the best performers.

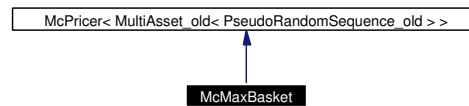
Public Member Functions

- **McHimalaya** (const std::vector< double > &underlying, const **Array** ÷ndYield, const **Matrix** &covariance, Rate riskFreeRate, double strike, const std::vector< Time > ×, bool antitheticVariance, long seed=0)

9.261 McMaxBasket Class Reference

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

Inheritance diagram for McMaxBasket:



9.261.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

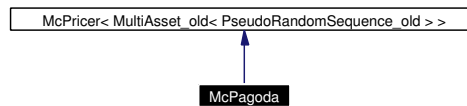
Public Member Functions

- **McMaxBasket** (const std::vector< double > &underlying, const **Array** ÷ndYield, const **Matrix** &covariance, Rate riskFreeRate, double residualTime, bool antitheticVariance, long seed=0)

9.262 McPagoda Class Reference

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

Inheritance diagram for McPagoda:



9.262.1 Detailed Description

roofed Asian option

Given a certain portfolio of assets at the end of the period it is returned the minimum of a given roof and a certain fraction of the positive portfolio performance. If the performance of the portfolio is below then the payoff is null.

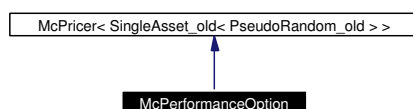
Public Member Functions

- **McPagoda** (const std::vector< double > &portfolio, double fraction, double roof, const **Array** ÷ndYield, const **Matrix** &covariance, Rate riskFreeRate, const std::vector< Time > ×, bool antithetic, long seed=0)

9.263 McPerformanceOption Class Reference

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

Inheritance diagram for McPerformanceOption:



9.263.1 Detailed Description

Performance option computed using Monte Carlo simulation.

A performance option is a variant of a cliquet option: the payoff of each forward-starting (a.k.a. deferred strike) options is \$ $\max(S/X - 1)$ \$.

Public Member Functions

- **McPerformanceOption** (Option::Type type, double underlying, double moneyiness, const std::vector< Spread > ÷ndYield, const std::vector< Rate > &riskFreeRate, const std::vector< Time > ×, const std::vector< double > &volatility, bool antitheticVariance, long seed=0)

9.264 McPricer Class Template Reference

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

9.264.1 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` (p. 438) 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

- double **value** (double tolerance, Size maxSample=QL_MAX_INT) const
add samples until the required tolerance is reached
- double **valueWithSamples** (Size samples) const
simulate a fixed number of samples
- double **errorEstimate** () const
error Estimated of the samples simulated so far
- const S & **sampleAccumulator** (void) const
access to the sample accumulator for more statistics

Protected Attributes

- `Handle< MonteCarloModel< MC, S > > mcModel_`

Static Protected Attributes

- const Size **minSample_** = 1023

9.265 McSimulation Class Template Reference

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

9.265.1 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

- double **value** (double tolerance, Size maxSample=QL_MAX_INT) const
add samples until the required tolerance is reached
- double **valueWithSamples** (Size samples) const
simulate a fixed number of samples
- double **errorEstimate** () const
error estimated using the samples simulated so far
- const stats_type & **sampleAccumulator** (void) const
access to the sample accumulator for richer statistics

Protected Member Functions

- **McSimulation** (bool antitheticVariate, bool controlVariate)
- virtual **Handle**< path_pricer_type > **pathPricer** () const=0
- virtual **Handle**< path_pricer_type > **controlPathPricer** () const
- virtual **Handle**< PricingEngine > **controlPricingEngine** () const
- virtual **Handle**< path_generator_type > **pathGenerator** () const=0
- virtual **TimeGrid** **timeGrid** () const=0

Protected Attributes

- **Handle**< **MonteCarloModel**< MC, S > > **mcModel_**
- bool **antitheticVariate_**
- bool **controlVariate_**

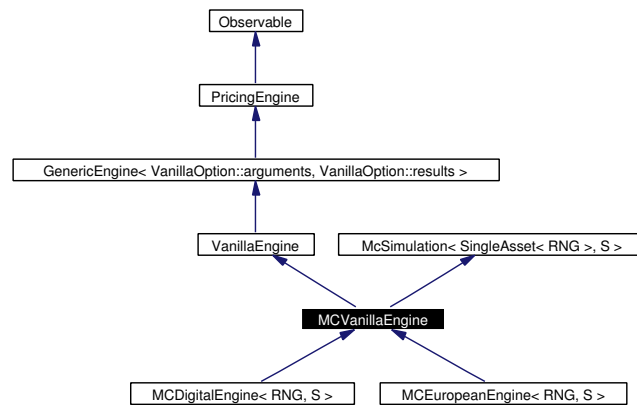
Static Protected Attributes

- const Size **minSample_** = 1023

9.266 MCVanillaEngine Class Template Reference

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

Inheritance diagram for MCVanillaEngine:



9.266.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCVanillaEngine< RNG, S >
```

Pricing engine for vanilla option using Monte Carlo simulation.

Public Member Functions

- `void calculate () const`

Protected Types

- `typedef McSimulation< SingleAsset< RNG >, S >::path_generator_type path_generator_type`
- `typedef McSimulation< SingleAsset< RNG >, S >::path_pricer_type path_pricer_type`
- `typedef McSimulation< SingleAsset< RNG >, S >::stats_type stats_type`

Protected Member Functions

- **MCVanillaEngine** (Size maxTimeStepsPerYear, bool antitheticVariate=false, bool controlVariate=false, Size requiredSamples=Null< int >(), double requiredTolerance=Null< double >(), Size maxSamples=Null< int >(), long seed=0)
- **Handle**< path_generator_type > **pathGenerator** () const

Protected Attributes

- Size **maxTimeStepsPerYear_**

- Size `requiredSamples_`
- Size `maxSamples_`
- double `requiredTolerance_`
- long `seed_`

9.267 MersenneTwisterUniformRng Class Reference

```
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

9.267.1 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>

Public Types

- typedef **Sample**< double > **sample_type**

Public Member Functions

- **MersenneTwisterUniformRng** (unsigned long seed=0)
- **MersenneTwisterUniformRng** (const std::vector< unsigned long > &seeds)
- sample_type **next** () const
- unsigned long **nextInt32** () const

return a random number on $[0, 0xffffffff]$ -interval

9.267.2 Constructor & Destructor Documentation

9.267.2.1 MersenneTwisterUniformRng (unsigned long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

9.267.3 Member Function Documentation

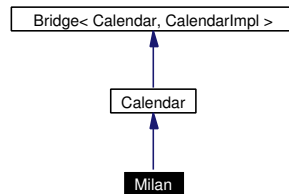
9.267.3.1 sample_type next () const

returns a sample with weight 1.0 containing a random number on (0.0, 1.0)-real-interval

9.268 Milan Class Reference

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

Inheritance diagram for Milan:



9.268.1 Detailed Description

Milan calendar

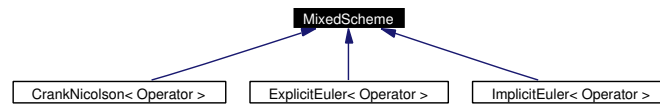
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Easter Monday
- Liberation Day, April 25th
- Labour Day, May 1st
- Republic Day, June 2nd (since 2000)
- Assumption, August 15th
- All Saint's Day, November 1st
- Immaculate Conception, December 8th
- Christmas, December 25th
- St. Stephen, December 26th

9.269 MixedScheme Class Template Reference

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

Inheritance diagram for MixedScheme:



9.269.1 Detailed Description

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

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

See sect. **The finite differences framework**(p. 37) for details on the method.

In this implementation, the passed operator must be derived from either `TimeConstantOperator` or `TimeDependentOperator`. Also, it must implement at least the following interface:

```
typedef ... arrayType;

// copy constructor/assignment
// (these will be provided by the compiler if none is defined)
Operator(const Operator&);
Operator& operator=(const Operator&);

// inspectors
Size size();

// modifiers
void setTime(Time t);

// operator interface
arrayType applyTo(const arrayType&);
arrayType solveFor(const arrayType&);
static Operator identity(Size size);

// operator algebra
Operator operator*(double, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Warning:

The differential operator must be linear for this evolver to work.

Todo

a) derive variable theta schemes b) introduce multi time-level schemes.

Protected Types

- typedef `Operator::arrayType` **arrayType**
- typedef `Operator` **operatorType**
- typedef `BoundaryCondition< Operator >` **bcType**

Protected Member Functions

- **MixedScheme** (const Operator &L, double theta, const std::vector< **Handle**< bcType > > &bcs)
- void **step** (arrayType &a, Time t)
- void **setStep** (Time dt)

Protected Attributes

- Operator **L_**
- Operator **I_**
- Operator **explicitPart_**
- Operator **implicitPart_**
- Time **dt_**
- double **theta_**
- std::vector< **Handle**< bcType > > **bcs_**

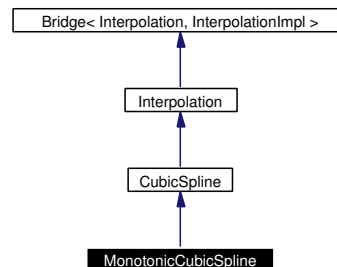
Friends

- class **FiniteDifferenceModel**< **MixedScheme**< Operator > >

9.270 MonotonicCubicSpline Class Reference

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

Inheritance diagram for MonotonicCubicSpline:



9.270.1 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, double leftConditionValue, CubicSpline::BoundaryCondition rightCondition, double rightConditionValue)`

9.270.2 Constructor & Destructor Documentation

- 9.270.2.1 `MonotonicCubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, CubicSpline::BoundaryCondition leftCondition, double leftConditionValue, CubicSpline::BoundaryCondition rightCondition, double rightConditionValue)`

Precondition:

the x values must be sorted.

9.271 MonteCarloModel Class Template Reference

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

9.271.1 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 `mctraits.hpp` (p. 798) 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.

Public Types

- typedef mc_traits::rsg_type **rsg_type**
- typedef mc_traits::path_generator_type **path_generator_type**
- typedef mc_traits::path_pricer_type **path_pricer_type**
- typedef path_generator_type::sample_type **sample_type**
- typedef path_pricer_type::result_type **result_type**
- typedef stats_traits **stats_type**

Public Member Functions

- **MonteCarloModel** (const **Handle**< path_generator_type > &pathGenerator, const **Handle**< path_pricer_type > &pathPricer, const stats_type &sampleAccumulator, bool antitheticVariate, const **Handle**< path_pricer_type > &cvPathPricer=**Handle**< path_pricer_type >(), result_type cvOptionValue=result_type())
- void **addSamples** (Size samples)
- const stats_type & **sampleAccumulator** (void) const

9.272 MoreGreeks Class Reference

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

Inheritance diagram for MoreGreeks:



9.272.1 Detailed Description

more additional option results

Public Member Functions

- void **reset** ()

Public Attributes

- double **itmCashProbability**
- double **deltaForward**
- double **elasticity**
- double **thetaPerDay**
- double **strikeSensitivity**

9.273 MoroInverseCumulativeNormal Class Reference

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

9.273.1 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**(p. [378](#))

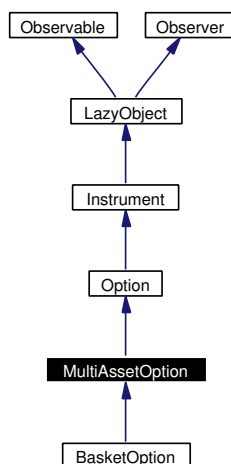
Public Member Functions

- **MoroInverseCumulativeNormal** (double average=0.0, double sigma=1.0)
- double **operator()** (double x) const

9.274 MultiAssetOption Class Reference

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

Inheritance diagram for MultiAssetOption:



9.274.1 Detailed Description

Base class for options on multiple assets.

Public Member Functions

- **MultiAssetOption** (const std::vector< **Handle**< BlackScholesStochasticProcess > > &stoch-
Procs, const **Handle**< Payoff > &payoff, const **Handle**< Exercise > &exercise, const **Matrix**
&correlation, const **Handle**< PricingEngine > &engine=**Handle**< PricingEngine >())
- void **setupArguments** (**Arguments** *) const

Instrument interface

- bool **isExpired** () const
returns whether the instrument is still tradable.

greeks

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

Protected Member Functions

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

Protected Attributes

- double **delta_**
- double **gamma_**
- double **theta_**
- double **vega_**
- double **rho_**
- double **dividendRho_**
- std::vector< **Handle**< BlackScholesStochasticProcess > > **blackScholesProcesses_**
- **Matrix correlation_**

9.274.2 Member Function Documentation

9.274.2.1 void setupArguments (Arguments *) const [virtual]

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

Reimplemented from **Instrument** (p. [368](#)).

Reimplemented in **BasketOption** (p. [140](#)).

9.274.2.2 void setupExpired () const [protected, virtual]

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

Reimplemented from **Instrument** (p. [369](#)).

9.274.2.3 void performCalculations () const [protected, virtual]

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

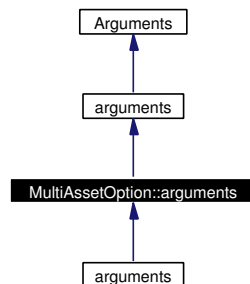
Reimplemented from **Instrument** (p. [369](#)).

Reimplemented in **BasketOption** (p. [141](#)).

9.275 MultiAssetOption::arguments Class Reference

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

Inheritance diagram for MultiAssetOption::arguments:



9.275.1 Detailed Description

Arguments for multi-asset option calculation

Public Member Functions

- `void validate () const`

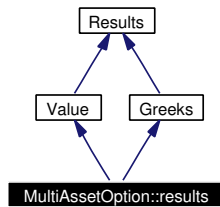
Public Attributes

- `std::vector< Handle< BlackScholesStochasticProcess > > blackScholesProcesses`
- `Matrix correlation`

9.276 MultiAssetOption::results Class Reference

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

Inheritance diagram for MultiAssetOption::results:



9.276.1 Detailed Description

Results from multi-asset option calculation

Public Member Functions

- `void reset ()`

9.277 MultiPath Class Reference

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

9.277.1 Detailed Description

multiple random walk

MultiPath(p. 455) contains the list of variations for each asset,

$$\log \frac{Y_{i+1}^j}{Y_i^j} \text{ for } i = 0, \dots, n-1 \quad \text{and} \quad j = 0, \dots, m-1$$

where Y_i^j is the value of the underlying j at discretized time t_i . The first index refers to the underlying, the second to the time position **MultiPath**(p. 455)[j,i]

Todo

- a) make it time-aware b) rename it as MultiAssetPath

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)

9.278 MultiPathGenerator Class Template Reference

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

9.278.1 Detailed Description

```
template<class GSG> class QuantLib::MultiPathGenerator< GSG >
```

Generates a multipath from a random number generator.

MultiPathGenerator<RAG> is a class that returns a random multi path. RAG is a sample generator which returns a random array. It must have the minimal interface:

```
RAG{
    RAG();
    RAG(Matrix& covariance,
        long seed);
    Sample<Array> next();
};
```

Todo

why store correlation **Matrix**([p. 418](#)) rather than covariance?

Public Types

- typedef **Sample< MultiPath > sample_type**

Public Member Functions

- **MultiPathGenerator** (const std::vector< **Handle< DiffusionProcess > >** &diffusionProcs, const **Matrix** &correlation, const **TimeGrid** &timeGrid, GSG generator, bool brownian-Bridge)
- const sample_type & **next** () const
- const sample_type & **antithetic** () const

9.279 MultiPathGenerator_old Class Template Reference

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

9.279.1 Detailed Description

```
template<class RAG> class QuantLib::MultiPathGenerator_old< RAG >
```

Generates a multipath from a random number generator.

MultiPathGenerator_old<RAG> is a class that returns a random multi path. RAG is a sample generator which returns a random array. It must have the minimal interface:

```
RAG{  
    RAG();  
    RAG(Matrix& covariance,  
        long seed);  
    Sample<Array> next();  
};
```

Public Types

- typedef `Sample< MultiPath >` `sample_type`

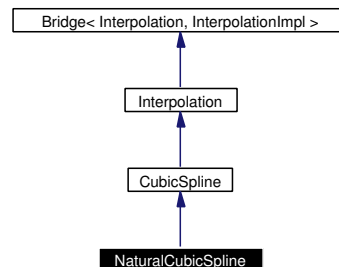
Public Member Functions

- **MultiPathGenerator_old** (const **Array** &drifts, const **Matrix** &covariance, Time length, Size timeSteps, long seed)
- **MultiPathGenerator_old** (const **Array** &drifts, const **Matrix** &covariance, const **TimeGrid** ×, long seed=0)
- const `sample_type` & **next** () const
- const `sample_type` & **antithetic** () const

9.280 NaturalCubicSpline Class Reference

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

Inheritance diagram for NaturalCubicSpline:



9.280.1 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)`

9.280.2 Constructor & Destructor Documentation

9.280.2.1 NaturalCubicSpline (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

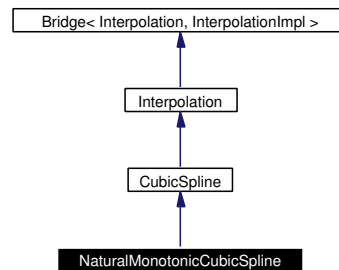
Precondition:

the *x* values must be sorted.

9.281 NaturalMonotonicCubicSpline Class Reference

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

Inheritance diagram for NaturalMonotonicCubicSpline:



9.281.1 Detailed Description

Natural cubic spline with monotonicity constraint.

Public Member Functions

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

9.281.2 Constructor & Destructor Documentation

9.281.2.1 NaturalMonotonicCubicSpline (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

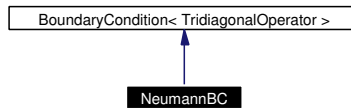
Precondition:

the *x* values must be sorted.

9.282 NeumannBC Class Reference

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

Inheritance diagram for NeumannBC:



9.282.1 Detailed Description

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

Warning:

The value passed must not be the value of the derivative. Instead, it must be comprehensive of the grid step between the first two points—i.e., it must be the difference between $f[0]$ and $f[1]$.

Todo

generalize to time-dependent conditions.

Public Member Functions

- **NeumannBC** (double value, Side side)
- void **applyBeforeApplying** (**TridiagonalOperator** &) const
- void **applyAfterApplying** (**Array** &) const
- void **applyBeforeSolving** (**TridiagonalOperator** &, **Array** &rhs) const
- void **applyAfterSolving** (**Array** &) const
- void **setTime** (Time t)

9.282.2 Member Function Documentation

9.282.2.1 void setTime (Time t) [virtual]

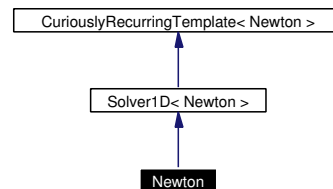
This method sets the current time for time-dependent boundary conditions.

Implements **BoundaryCondition**< **TridiagonalOperator** > (p. 173).

9.283 Newton Class Reference

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

Inheritance diagram for Newton:



9.283.1 Detailed Description

Newton 1-D solver

Note:

This solver requires that the passed function object implement a method `double derivative(double)`.

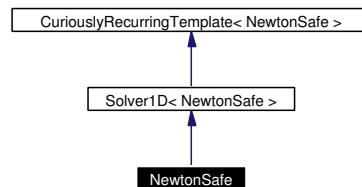
Public Member Functions

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

9.284 NewtonSafe Class Reference

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

Inheritance diagram for NewtonSafe:



9.284.1 Detailed Description

safe Newton 1-D solver

Note:

This solver requires that the passed function object implement a method `double derivative(double)`.

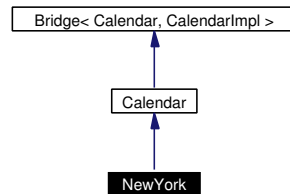
Public Member Functions

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

9.285 NewYork Class Reference

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

Inheritance diagram for NewYork:



9.285.1 Detailed Description

New York calendar.

Holidays:

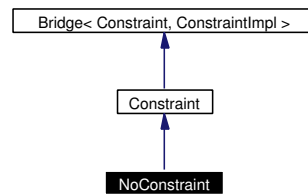
- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday, or to Friday if on Saturday)
- Martin Luther King's birthday, third Monday in January
- Washington's birthday, third Monday in February
- Good Friday
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

Data from <http://www.nyse.com>

9.286 NoConstraint Class Reference

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

Inheritance diagram for NoConstraint:



9.286.1 Detailed Description

No constraint.

9.287 NonLinearLeastSquare Class Reference

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

9.287.1 Detailed Description

Non-linear least-square method.

Using a given optimization algorithm (default is conjugate gradient),

$\min \{ r(x) : x \text{ in } \mathbb{R}^n \}$

where $r(x) = \|f(x)\|^2$ the euclidian norm of $f(x)$ for some vector-valued function f from \mathbb{R}^n to \mathbb{R}^m $f = (f_1, \dots, f_m)$ with $f_i(x) = b_i - \phi_i(x, t_i)$ where b_i is the vector of target data and ϕ_i is a scalar function.

Assuming the differentiability of f , the gradient of r is define by $\text{grad } r(x) = f'(x)^t \cdot f(x)$

Array(p. 126) vector class has the requirement of the previous class **Handle**(p. 339) class is need to manage pointer to optimization method

Public Member Functions

- **NonLinearLeastSquare** (**Constraint** &c, double accuracy=1e-4, int maxiter=100)
Default constructor.
- **NonLinearLeastSquare** (**Constraint** &c, double accuracy, int maxiter, **Handle**<**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
- double **residualNorm** ()
return the least square residual norm
- double **lastValue** ()
return last function value
- int **exitFlag** ()
return exit flag
- int **iterationsNumber** ()
return the performed number of iterations

9.288 NormalDistribution Class Reference

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

9.288.1 Detailed Description

Normal distribution function.

formula here ... Given x it returns its probability in a Gaussian normal distribution. It provides the first derivative too.

Public Member Functions

- **NormalDistribution** (double average=0.0, double sigma=1.0)
- double **operator()** (double x) const
- double **derivative** (double x) const

9.289 Null Class Template Reference

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

9.289.1 Detailed Description

```
template<class Type> class QuantLib::Null< Type >
```

template class providing a null value for a given type.

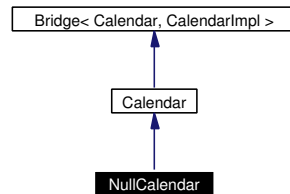
Public Member Functions

- `operator Type () const`

9.290 NullCalendar Class Reference

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

Inheritance diagram for NullCalendar:



9.290.1 Detailed Description

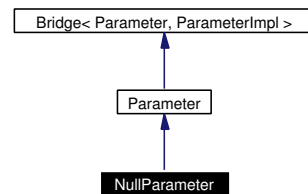
Calendar for reproducing theoretical calculations.

This calendar has no holidays. It ensures that dates at whole-month distances have the same day of month.

9.291 NullParameter Class Reference

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

Inheritance diagram for NullParameter:



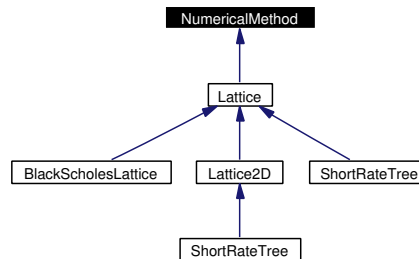
9.291.1 Detailed Description

Parameter which is always zero $a(t) = 0$

9.292 NumericalMethod Class Reference

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

Inheritance diagram for NumericalMethod:



9.292.1 Detailed Description

Numerical method (tree, finite-differences) base class.

Public Member Functions

- NumericalMethod (const TimeGrid &timeGrid)
- const TimeGrid & timeGrid () const
- virtual void initialize (const Handle< DiscretizedAsset > &, Time time) const=0
- virtual void rollback (const Handle< DiscretizedAsset > &, Time to) const=0
- virtual void rollAlmostBack (const Handle< DiscretizedAsset > &, Time to) const=0

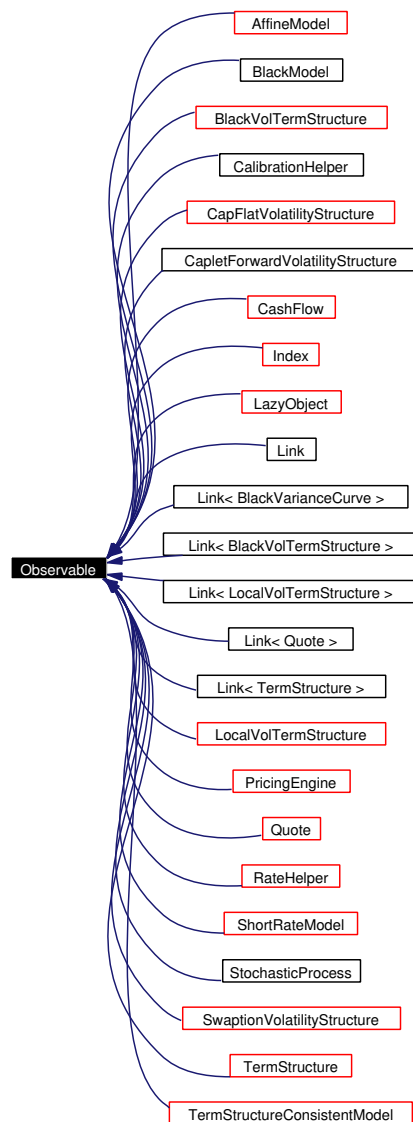
Protected Attributes

- TimeGrid t_

9.293 Observable Class Reference

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

Inheritance diagram for Observable:



9.293.1 Detailed Description

Object that notifies its changes to a set of observables.

Public Member Functions

- void **notifyObservers** ()

Friends

- class **Observer**

9.293.2 Member Function Documentation

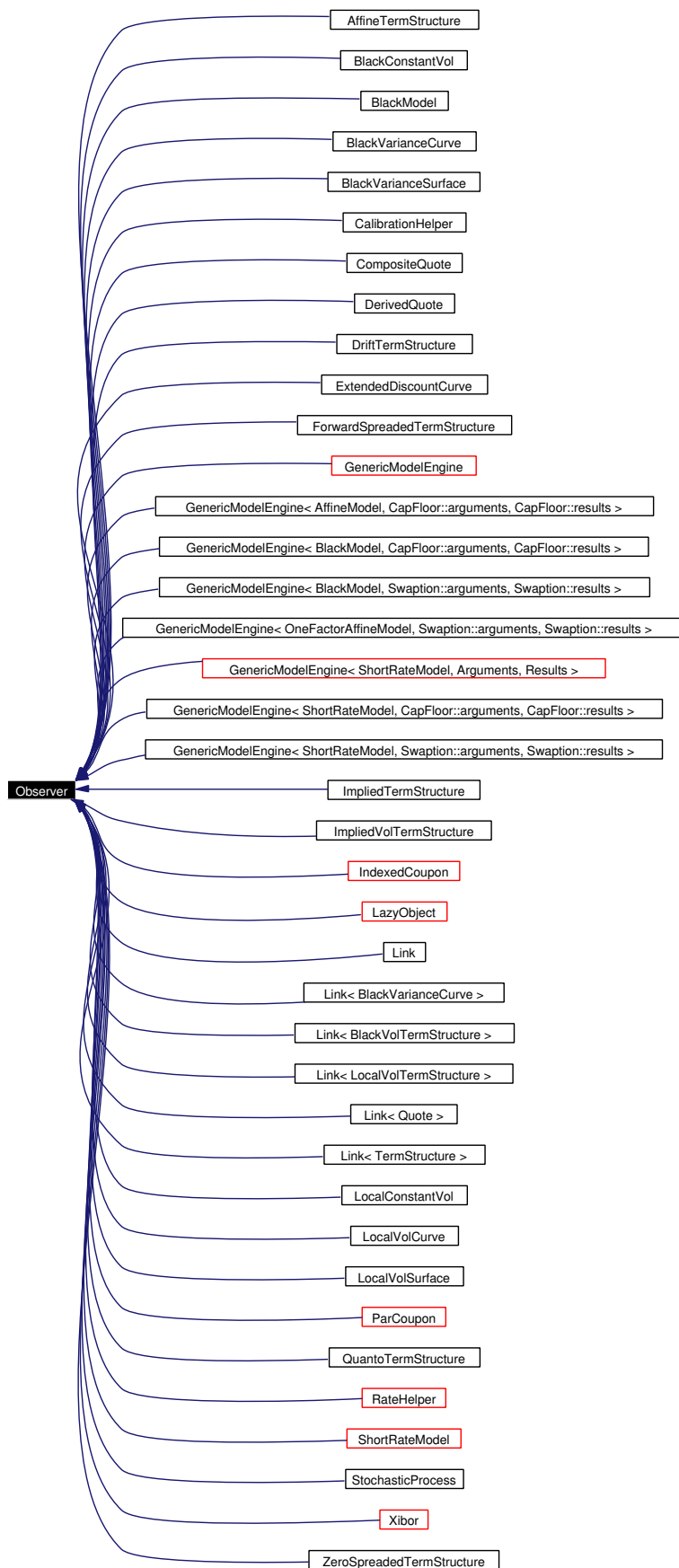
9.293.2.1 void notifyObservers ()

This method should be called at the end of non-const methods or when the programmer desires to notify any changes.

9.294 Observer Class Reference

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

Inheritance diagram for Observer:



9.294.1 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 **Handle**< T > &h)
- template<class T> void **unregisterWith** (const **Handle**< T > &h)
- virtual void **update** ()=0

9.294.2 Member Function Documentation

9.294.2.1 virtual void update () [pure virtual]

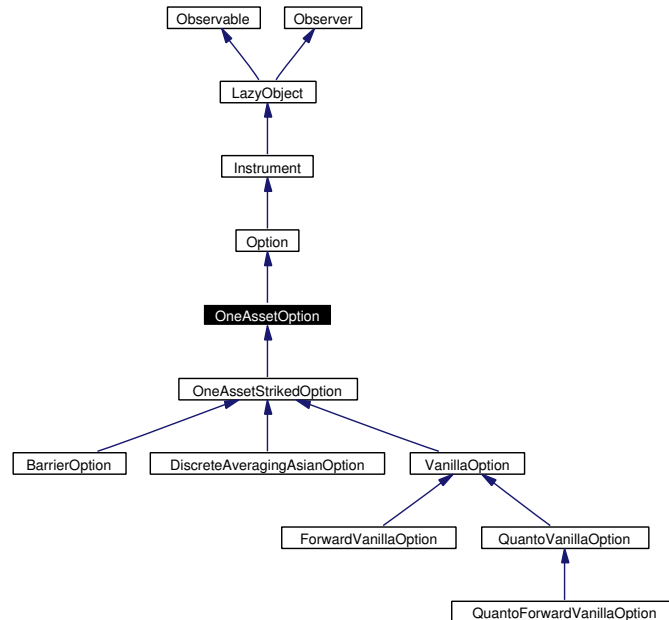
This method must be implemented in derived classes. An instance of **Observer** does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implemented in **IndexedCoupon** (p. 365), **ParCoupon** (p. 500), **Xibor** (p. 632), **DerivedQuote** (p. 245), **CompositeQuote** (p. 212), **LazyObject** (p. 392), **BlackModel** (p. 159), **GenericModelEngine** (p. 333), **LatticeShortRateModelEngine** (p. 389), **Link** (p. 404), **CalibrationHelper** (p. 190), **ShortRateModel** (p. 551), **StochasticProcess** (p. 573), **AffineTermStructure** (p. 114), **DriftTermStructure** (p. 270), **ExtendedDiscountCurve** (p. 289), **ForwardSpreadedTermStructure** (p. 314), **ImpliedTermStructure** (p. 356), **QuantoTermStructure** (p. 529), **RateHelper** (p. 537), **ZeroSpreadedTermStructure** (p. 637), **BlackConstantVol** (p. 156), **BlackVarianceCurve** (p. 165), **BlackVarianceSurface** (p. 167), **ImpliedVolTermStructure** (p. 358), **LocalConstantVol** (p. 407), **LocalVolCurve** (p. 409), **LocalVolSurface** (p. 411), **GenericModelEngine**< **ShortRateModel**, **Arguments**, **Results** > (p. 333), **GenericModelEngine**< **BlackModel**, **CapFloor::arguments**, **CapFloor::results** > (p. 333), **GenericModelEngine**< **OneFactorAffineModel**, **Swaption::arguments**, **Swaption::results** > (p. 333), **GenericModelEngine**< **AffineModel**, **CapFloor::arguments**, **CapFloor::results** > (p. 333), **GenericModelEngine**< **ShortRateModel**, **CapFloor::arguments**, **CapFloor::results** > (p. 333), **GenericModelEngine**< **BlackModel**, **Swaption::arguments**, **Swaption::results** > (p. 333), **GenericModelEngine**< **ShortRateModel**, **Swaption::arguments**, **Swaption::results** > (p. 333), **LatticeShortRateModelEngine**< **CapFloor::arguments**, **CapFloor::results** > (p. 389), **LatticeShortRateModelEngine**< **Swaption::arguments**, **Swaption::results** > (p. 389), **Link**< **LocalVolTermStructure** > (p. 404), **Link**< **BlackVarianceCurve** > (p. 404), **Link**< **BlackVolTermStructure** > (p. 404), **Link**< **TermStructure** > (p. 404), and **Link**< **Quote** > (p. 404).

9.295 OneAssetOption Class Reference

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

Inheritance diagram for OneAssetOption:



9.295.1 Detailed Description

Base class for options on a single asset.

Public Member Functions

- **OneAssetOption** (const **Handle**< BlackScholesStochasticProcess > &stochProc, const **Handle**< Payoff > &payoff, const **Handle**< Exercise > &exercise, const **Handle**< PricingEngine > &engine=**Handle**< PricingEngine >())
- double **impliedVolatility** (double price, double accuracy=1.0e-4, Size maxEvaluations=100, double minVol=QL_MIN_VOLATILITY, double maxVol=QL_MAX_VOLATILITY) const
- void **setupArguments** (**Arguments** *) const

Instrument interface

- bool **isExpired** () const
returns whether the instrument is still tradable.

Greeks

- double **delta** () const
- double **deltaForward** () const
- double **elasticity** () const

- double **gamma** () const
- double **theta** () const
- double **thetaPerDay** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const
- double **itmCashProbability** () const

Protected Member Functions

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

Protected Attributes

- double **delta_**
- double **deltaForward_**
- double **elasticity_**
- double **gamma_**
- double **theta_**
- double **thetaPerDay_**
- double **vega_**
- double **rho_**
- double **dividendRho_**
- double **itmCashProbability_**
- **Handle**< BlackScholesStochasticProcess > **blackScholesProcess_**

9.295.2 Member Function Documentation

9.295.2.1 double **impliedVolatility** (double *price*, double *accuracy* = 1.0e-4, Size *maxEvaluations* = 100, double *minVol* = QL_MIN_VOLATILITY, double *maxVol* = QL_MAX_VOLATILITY) const

Warning:

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.

Bug

run-time crashes are possible with the Borland compiler

9.295.2.2 void setupArguments (Arguments *) const [virtual]

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

Reimplemented from **Instrument** (p. 368).

Reimplemented in **DiscreteAveragingAsianOption** (p. 256), **BarrierOption** (p. 137), **Forward-VanillaOption** (p. 316), **OneAssetStrikedOption** (p. 482), **QuantoForwardVanillaOption** (p. 525), and **QuantoVanillaOption** (p. 531).

9.295.2.3 void setupExpired () const [protected, virtual]

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

Reimplemented from **Instrument** (p. 369).

Reimplemented in **QuantoVanillaOption** (p. 531).

9.295.2.4 void performCalculations () const [protected, virtual]

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

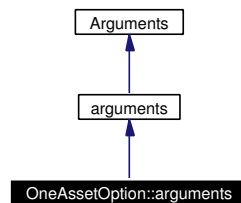
Reimplemented from **Instrument** (p. 369).

Reimplemented in **DiscreteAveragingAsianOption** (p. 256), **BarrierOption** (p. 137), **Forward-VanillaOption** (p. 316), **OneAssetStrikedOption** (p. 482), **QuantoVanillaOption** (p. 531), and **VanillaOption** (p. 624).

9.296 OneAssetOption::arguments Class Reference

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

Inheritance diagram for OneAssetOption::arguments:



9.296.1 Detailed Description

Arguments for single-asset option calculation

Public Member Functions

- void **validate** () const

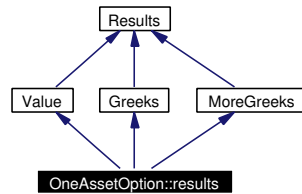
Public Attributes

- **Handle**< BlackScholesStochasticProcess > **blackScholesProcess**

9.297 OneAssetOption::results Class Reference

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

Inheritance diagram for OneAssetOption::results:



9.297.1 Detailed Description

Results from single-asset option calculation

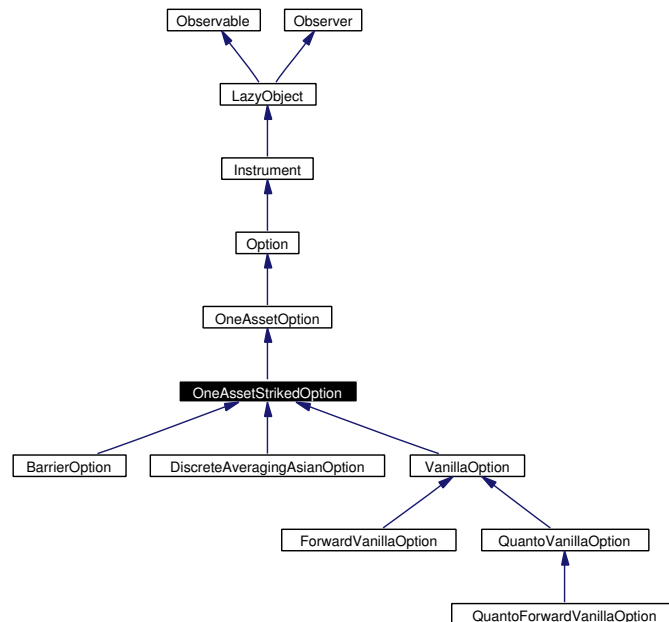
Public Member Functions

- `void reset ()`

9.298 OneAssetStrikedOption Class Reference

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

Inheritance diagram for OneAssetStrikedOption:



9.298.1 Detailed Description

Base class for options on a single asset with striked payoff.

Public Member Functions

- **OneAssetStrikedOption** (const **Handle**< **BlackScholesStochasticProcess** > &stochProc, const **Handle**< **StrikedTypePayoff** > &payoff, const **Handle**< **Exercise** > &exercise, const **Handle**< **PricingEngine** > &engine=**Handle**< **PricingEngine** >())
- void **setupArguments** (**Arguments** *) const

greeks

- double **strikeSensitivity** () const

Protected Member Functions

- void **performCalculations** () const

Protected Attributes

- double **strikeSensitivity_**

9.298.2 Member Function Documentation

9.298.2.1 `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** (p. 478).

Reimplemented in **DiscreteAveragingAsianOption** (p. 256), **BarrierOption** (p. 137), **Forward-VanillaOption** (p. 316), **QuantoForwardVanillaOption** (p. 525), and **QuantoVanillaOption** (p. 531).

9.298.2.2 `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.

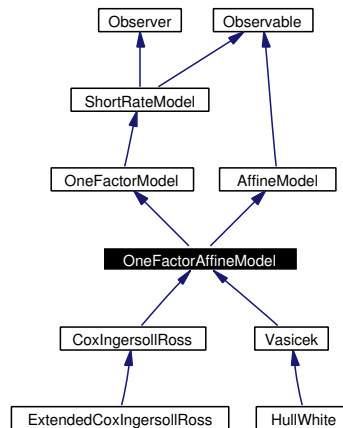
Reimplemented from **OneAssetOption** (p. 478).

Reimplemented in **DiscreteAveragingAsianOption** (p. 256), **BarrierOption** (p. 137), **Forward-VanillaOption** (p. 316), **QuantoVanillaOption** (p. 531), and **VanillaOption** (p. 624).

9.299 OneFactorAffineModel Class Reference

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

Inheritance diagram for OneFactorAffineModel:



9.299.1 Detailed Description

Single-factor affine base class.

Single-factor models with an analytical formula for discount bonds should inherit from this class. They must then implement the functions $A(t, T)$ and $B(t, T)$ such that

$$P(t, T, r_t) = A(t, T)e^{-B(t, T)r_t}.$$

Public Member Functions

- **OneFactorAffineModel** (Size nArguments)
- double **discountBond** (Time now, Time maturity, Rate rate) const
- DiscountFactor **discount** (Time t) const

Implied discount curve.

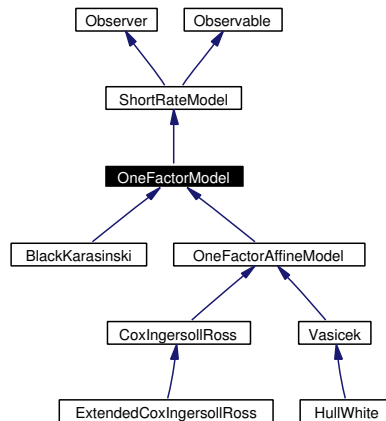
Protected Member Functions

- virtual double **A** (Time t, Time T) const=0
- virtual double **B** (Time t, Time T) const=0

9.300 OneFactorModel Class Reference

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

Inheritance diagram for OneFactorModel:



9.300.1 Detailed Description

Single-factor short-rate model abstract class.

Public Member Functions

- **OneFactorModel** (Size nArguments)
- virtual **Handle< ShortRateDynamics > dynamics** () const=0
returns the short-rate dynamics
- virtual **Handle< Lattice > tree** (const **TimeGrid** &grid) const
Return by default a trinomial recombining tree.

9.301 OneFactorModel::ShortRateDynamics Class Reference

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

9.301.1 Detailed Description

Base class describing the short-rate dynamics.

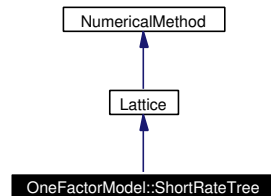
Public Member Functions

- **ShortRateDynamics** (const **Handle**< **DiffusionProcess** > &process)
- virtual double **variable** (Time t, Rate r) const=0
Compute state variable from short rate.
- virtual Rate **shortRate** (Time t, double variable) const=0
Compute short rate from state variable.
- const **Handle**< **DiffusionProcess** > & **process** ()
Returns the risk-neutral dynamics of the state variable.

9.302 OneFactorModel::ShortRateTree Class Reference

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

Inheritance diagram for OneFactorModel::ShortRateTree:



9.302.1 Detailed Description

Recombining trinomial tree discretizing the state variable.

Public Member Functions

- **ShortRateTree** (const **Handle**< **Tree** > &tree, const **Handle**< **ShortRateDynamics** > &dynamics, const **TimeGrid** &timeGrid)
Plain tree build-up from short-rate dynamics.
- **ShortRateTree** (const **Handle**< **Tree** > &tree, const **Handle**< **ShortRateDynamics** > &dynamics, const **Handle**< **TermStructureFittingParameter::NumericalImpl** > &phi, const **TimeGrid** &timeGrid)
***Tree**(p. 607) build-up + numerical fitting to term-structure.*
- Size **size** (Size i) const
- DiscountFactor **discount** (Size i, Size index) const
Discount factor at time t_i and node indexed by index.

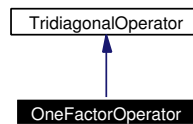
Protected Member Functions

- Size **descendant** (Size i, Size index, Size branch) const
***Tree**(p. 607) properties.*
- double **probability** (Size i, Size index, Size branch) const

9.303 OneFactorOperator Class Reference

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

Inheritance diagram for OneFactorOperator:



9.303.1 Detailed Description

Interest-rate single factor model differential operator.

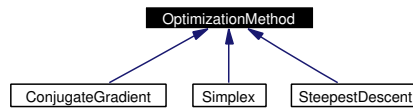
Public Member Functions

- **OneFactorOperator** (const **Array** &grid, const **Handle**< **OneFactorModel::ShortRateDynamics** > &)

9.304 OptimizationMethod Class Reference

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

Inheritance diagram for OptimizationMethod:



9.304.1 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.
- int & **iterationNumber** () const
current iteration number
- **EndCriteria** & **endCriteria** () const
optimization end criteria
- int & **functionEvaluation** () const
number of evaluation of cost function
- int & **gradientEvaluation** () const
number of evaluation of cost function gradient
- double & **functionValue** () const
value of cost function
- double & **gradientNormValue** () const
value of cost function gradient norm
- **Array** & **x** () const
current value of the local minimum
- **Array** & **searchDirection** () const
current value of the search direction
- virtual void **minimize** (const **Problem** &P) const=0
minimize the optimization problem P

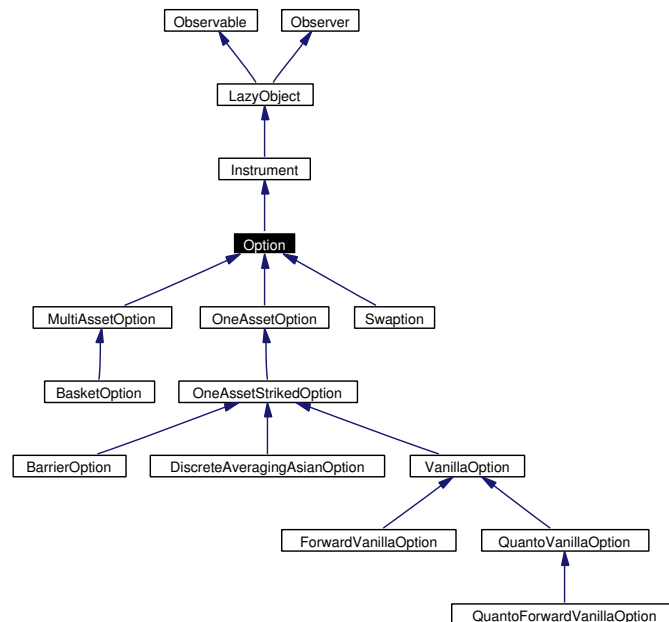
Protected Attributes

- **Array initialValue_**
initial value of unknowns
- **int iterationNumber_**
current iteration step in the Optimization process
- **EndCriteria endCriteria_**
optimization end criteria
- **int functionEvaluation_**
number of evaluation of cost function and its gradient
- **int gradientEvaluation_**
number of evaluation of cost function and its gradient
- **double functionValue_**
function and gradient norm values of the last step
- **double squaredNorm_**
function and gradient norm values of the last step
- **Array x_**
current values of the local minimum and the search direction
- **Array searchDirection_**
current values of the local minimum and the search direction

9.305 Option Class Reference

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

Inheritance diagram for Option:



9.305.1 Detailed Description

base option class

Public Types

- enum Type { Call, Put, Straddle }

Public Member Functions

- Option (const Handle< Payoff > &payoff, const Handle< Exercise > &exercise, const Handle< PricingEngine > &engine=Handle< PricingEngine >())

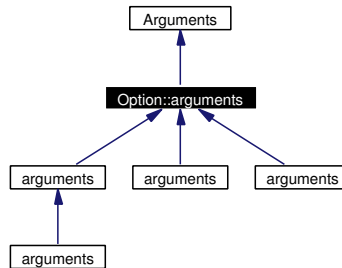
Protected Attributes

- Handle< Payoff > payoff_
- Handle< Exercise > exercise_

9.306 Option::arguments Class Reference

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

Inheritance diagram for Option::arguments:



9.306.1 Detailed Description

basic option arguments

Todo

- a) remove `std::vector<Time> stoppingTimes` b) how to handle strike-less option (asian average strike, forward, etc.)?

Public Member Functions

- `void validate () const`

Public Attributes

- `Handle< Payoff > payoff`
- `Handle< Exercise > exercise`
- `std::vector< Time > stoppingTimes`

9.307 OptionTypeFormatter Class Reference

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

9.307.1 Detailed Description

Formats option type for output.

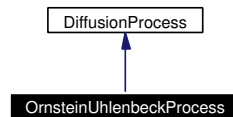
Static Public Member Functions

- `std::string toString (Option::Type type)`

9.308 OrnsteinUhlenbeckProcess Class Reference

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

Inheritance diagram for OrnsteinUhlenbeckProcess:



9.308.1 Detailed Description

Ornstein-Uhlenbeck process class.

This class describes the Ornstein-Uhlenbeck process governed by

$$dx = -ax_t dt + \sigma dW_t.$$

Public Member Functions

- **OrnsteinUhlenbeckProcess** (double speed, double vol, double x0=0.0)
- double **drift** (Time t, double x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- double **diffusion** (Time t, double x) const
- double **expectation** (Time t0, double x0, Time dt) const
returns the expectation of the process after a time interval
- double **variance** (Time t0, double x0, Time dt) const
returns the variance of the process after a time interval

9.308.2 Member Function Documentation

9.308.2.1 double diffusion (Time t, double x) const [virtual]

returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

Implements **DiffusionProcess** (p. 246).

9.308.2.2 double expectation (Time t0, double x0, Time dt) const [virtual]

returns the expectation of the process after a time interval

returns $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$. By default, it returns the Euler approximation defined by $x_0 + \mu(t_0, x_0)\Delta t$.

Reimplemented from **DiffusionProcess** (p. 246).

9.308.2.3 double variance (Time t_0 , double x_0 , Time dt) const [virtual]

returns the variance of the process after a time interval

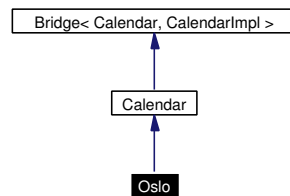
returns $\text{Var}(x_{t_0+\Delta t}|x_{t_0} = x_0)$. By default, it returns the Euler approximation defined by $\sigma(t_0, x_0)^2 \Delta t$.

Reimplemented from **DiffusionProcess** (p. [247](#)).

9.309 Oslo Class Reference

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

Inheritance diagram for Oslo:



9.309.1 Detailed Description

Oslo calendar

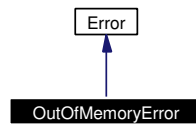
Holidays:

- 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

9.310 OutOfMemoryError Class Reference

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

Inheritance diagram for OutOfMemoryError:



9.310.1 Detailed Description

Specialized error.

Thrown upon failed allocation.

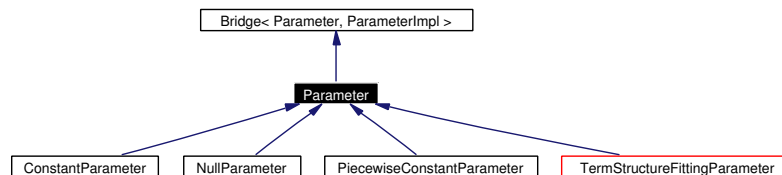
Public Member Functions

- **OutOfMemoryError** (const std::string &whatClass="unknown class")

9.311 Parameter Class Reference

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

Inheritance diagram for Parameter:



9.311.1 Detailed Description

Base class for model arguments.

Public Member Functions

- `const Array & params () const`
- `void setParam (Size i, double x)`
- `bool testParams (const Array ¶ms) const`
- `Size size () const`
- `double operator() (Time t) const`
- `const Handle< ParameterImpl > & implementation () const`

Protected Member Functions

- `Parameter (Size size, const Handle< ParameterImpl > &impl, const Constraint &constraint)`

Protected Attributes

- `Array params_`
- `Constraint constraint_`

9.312 ParameterImpl Class Reference

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

9.312.1 Detailed Description

Base class for model parameter implementation.

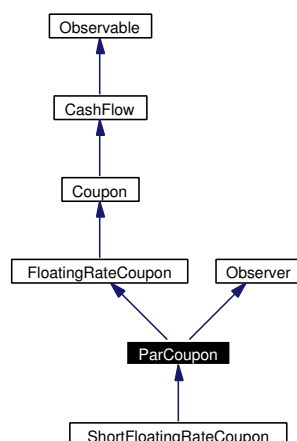
Public Member Functions

- virtual double **value** (const **Array** ¶ms, Time t) const=0

9.313 ParCoupon Class Reference

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

Inheritance diagram for ParCoupon:



9.313.1 Detailed Description

coupon at par on a term structure

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **ParCoupon** (double nominal, const **Date** &paymentDate, const **Handle**< **Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, Spread spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())

CashFlow interface

- double **amount** () const
returns the amount of the cash flow

Coupon interface

- **DayCounter** **dayCounter** () const
day counter for accrual calculation

FloatingRateCoupon interface

- **Rate** **fixing** () const

- **Date** `fixingDate () const`

Inspectors

- `const Handle< Xibor > & index () const`

Observer interface

- `void update ()`

Visitability

- `virtual void accept (AcyclicVisitor &)`

9.313.2 Member Function Documentation

9.313.2.1 `double amount () const` [virtual]

returns the amount of the cash flow

Note:

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

Implements **CashFlow** (p. [200](#)).

Reimplemented in **ShortFloatingRateCoupon** (p. [549](#)).

9.313.2.2 `void update ()` [virtual]

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

Implements **Observer** (p. [475](#)).

9.314 Path Class Reference

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

9.314.1 Detailed Description

single factor random walk path pricer

Todo

should **Path**(p. [501](#)) include the t=0.0 point? Alternatively all path pricers must be revisited.

Public Member Functions

- **Path** (const **TimeGrid** &timeGrid, const **Array** &drift=**Array**(), const **Array** &diffusion=**Array**())

inspectors

- double **operator**[] (int i) const
- Size **size** () const

read/write access to components

- const **TimeGrid** & **timeGrid** () const
- **TimeGrid** & **timeGrid** ()
- const **Array** & **drift** () const
- **Array** & **drift** ()
- const **Array** & **diffusion** () const
- **Array** & **diffusion** ()

9.315 PathGenerator Class Template Reference

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

9.315.1 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

Public Types

- typedef **Sample< Path > sample_type**

Public Member Functions

- **PathGenerator** (const **Handle< DiffusionProcess >** &diffProcess, Time length, Size timeSteps, const GSG &generator, bool brownianBridge)
- **PathGenerator** (const **Handle< DiffusionProcess >** &diffProcess, 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

9.316 PathGenerator_old Class Template Reference

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

9.316.1 Detailed Description

```
template<class RNG> class QuantLib::PathGenerator_old< RNG >
```

Generates random paths from a random number generator.

Deprecated

use `PathGenerator`(p. 502) instead

Public Types

- typedef `Sample< Path > sample_type`

Public Member Functions

- `PathGenerator_old` (double drift, double variance, Time length, Size timeSteps, long seed=0)
- `PathGenerator_old` (double drift, double variance, const `TimeGrid` ×, long seed=0)
- `PathGenerator_old` (const std::vector< double > &drift, const std::vector< double > &variance, const `TimeGrid` ×, long seed=0)

inspectors

- const `sample_type` & `next` () const
- const `sample_type` & `antithetic` () const
- Size `size` () const

9.316.2 Constructor & Destructor Documentation

9.316.2.1 `PathGenerator_old` (double *drift*, double *variance*, const `TimeGrid` & *times*, long *seed* = 0)

Warning:

the initial time is assumed to be zero and must **not** be included in the passed vector

9.317 PathPricer Class Template Reference

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

9.317.1 Detailed Description

```
template<class PathType, class ValueType = double> class QuantLib::PathPricer< PathType,
ValueType >
```

base class for path pricers

Given a path the value of an option is returned on that path.

Public Member Functions

- **PathPricer** (const **RelinkableHandle**< **TermStructure** > &riskFreeTS)
- virtual **ValueType** **operator()** (const **PathType** &path) const=0

Protected Attributes

- **RelinkableHandle**< **TermStructure** > **riskFreeTS_**

9.318 PathPricer_old Class Template Reference

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

9.318.1 Detailed Description

```
template<class PathType, class ValueType = double> class QuantLib::PathPricer_old< Path-  
Type, ValueType >
```

base class for path pricers

Given a path the value of an option is returned on that path.

Deprecated

use `PathPricer`(p. [504](#)) instead

Public Member Functions

- `PathPricer_old` (DiscountFactor discount, bool useAntitheticVariance)
- virtual ValueType `operator()` (const PathType &path) const=0

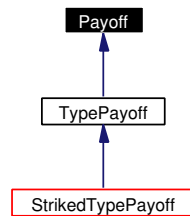
Protected Attributes

- DiscountFactor `discount_`
- bool `useAntitheticVariance_`

9.319 Payoff Class Reference

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

Inheritance diagram for Payoff:



9.319.1 Detailed Description

Base class for option payoffs.

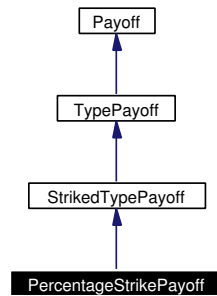
Public Member Functions

- virtual double **operator()** (double price) const=0

9.320 PercentageStrikePayoff Class Reference

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

Inheritance diagram for PercentageStrikePayoff:



9.320.1 Detailed Description

Payoff with strike expressed as percentage

Public Member Functions

- **PercentageStrikePayoff** (Option::Type type, double moneyness)
- double **operator()** (double price) const

9.321 PerformanceOption Class Reference

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

9.321.1 Detailed Description

Performance option.

A performance option is a variant of a cliquet option: the payoff of each forward-starting (a.k.a. deferred strike) options is $\$ \max(S/X - 1) \$$.

Public Member Functions

- **PerformanceOption** (Option::Type type, double underlying, double moneyness, const std::vector< Spread > ÷ndYield, const std::vector< Rate > &riskFreeRate, const std::vector< Time > ×, const std::vector< double > &volatility)
- double **value** () const
- double **delta** () const
- double **gamma** () const
- double **theta** () const
- double **vega** () const
- double **rho** () const
- double **dividendRho** () const

9.322 Period Class Reference

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

9.322.1 Detailed Description

Time period described by a number of a given time unit.

Public Member Functions

- **Period** (int n, TimeUnit units)
- int **length** () const
- TimeUnit **units** () const

Related Functions

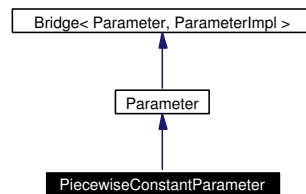
(Note that these are not member functions.)

- bool **operator<** (const **Period** &, const **Period** &)
- bool **operator==** (const **Period** &, const **Period** &)
- bool **operator!=** (const **Period** &, const **Period** &)
- bool **operator>** (const **Period** &, const **Period** &)
- bool **operator<=** (const **Period** &, const **Period** &)
- bool **operator>=** (const **Period** &, const **Period** &)

9.323 PiecewiseConstantParameter Class Reference

#include <ql/ShortRateModels/parameter.hpp>

Inheritance diagram for PiecewiseConstantParameter:



9.323.1 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

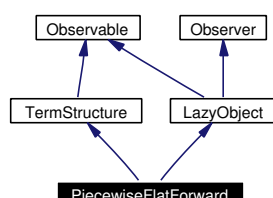
Public Member Functions

- `PiecewiseConstantParameter` (const std::vector< Time > ×)

9.324 PiecewiseFlatForward Class Reference

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

Inheritance diagram for PiecewiseFlatForward:



9.324.1 Detailed Description

Piecewise flat forward term structure.

This term structure is bootstrapped on a number of interest rate instruments which are passed as a vector of handles to **RateHelper**(p. 536) instances. Their maturities mark the boundaries of the flat forward segments.

The values of the forward rates for each segment are determined sequentially starting from the earliest period to the latest.

The value for each segment is chosen so that the instrument whose maturity marks the end of such segment is correctly repriced on the curve.

Rates are assumed to be annual continuous compounding.

Warning:

The bootstrapping algorithm will raise an exception if any two instruments have the same maturity date.

Public Member Functions

- **PiecewiseFlatForward** (const **Date** &todayDate, const **Date** &referenceDate, const std::vector< **Handle**< **RateHelper** > > &instruments, const **DayCounter** &dayCounter, double accuracy=1.0e-12)
- **PiecewiseFlatForward** (const **Date** &todayDate, const std::vector< **Date** > &dates, const std::vector< **Rate** > &forwards, const **DayCounter** &dayCounter)

TermStructure interface

- **DayCounter** dayCounter () const
the day counter used for date/time conversion
- **Date** todayDate () const
today's date
- **Date** referenceDate () const
the reference date, i.e., the date at which discount = 1

- `const std::vector< Date > & dates () const`
- `Date maxDate () const`
the latest date for which the curve can return rates
- `const std::vector< Time > & times () const`
- `Time maxTime () const`
the latest time for which the curve can return rates

Protected Member Functions

- Rate **zeroYieldImpl** (Time, bool extrapolate=false) const
zero-yield calculation
- DiscountFactor **discountImpl** (Time, bool extrapolate=false) const
discount calculation
- Rate **forwardImpl** (Time, bool extrapolate=false) const
instantaneous forward-rate calculation
- Rate **compoundForwardImpl** (Time t, int compFreq, bool extrapolate) const
compound forward-rate calculation

Friends

- class **FFObjFunction**

9.324.2 Constructor & Destructor Documentation

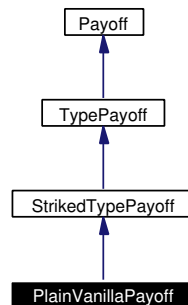
9.324.2.1 PiecewiseFlatForward (const Date & todaysDate, const std::vector< Date > & dates, const std::vector< Rate > & forwards, const DayCounter & dayCounter)

In this constructor, the first date must be the reference date of the curve, the other dates are the nodes of the term structure. The forward rate at index i is used in the period $t_{i-1} < t \leq t_i$. Therefore, forwards[0] is used only to compute the zero yield for $t = 0$.

9.325 PlainVanillaPayoff Class Reference

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

Inheritance diagram for PlainVanillaPayoff:



9.325.1 Detailed Description

Plain-vanilla payoff.

Public Member Functions

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

9.326 PoissonDistribution Class Reference

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

9.326.1 Detailed Description

Normal distribution function.

formula here ... Given an integer k it returns its probability in a Poisson distribution.

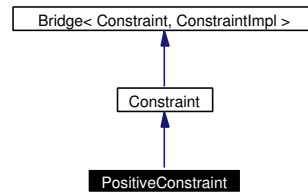
Public Member Functions

- **PoissonDistribution** (double mu)
- double **operator()** (unsigned long k) const

9.327 PositiveConstraint Class Reference

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

Inheritance diagram for PositiveConstraint:



9.327.1 Detailed Description

Constraint imposing positivity to all arguments

9.328 PostconditionNotSatisfiedError Class Reference

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

Inheritance diagram for PostconditionNotSatisfiedError:



9.328.1 Detailed Description

Specialized error.

Thrown upon an unsatisfied postcondition.

Public Member Functions

- `PostconditionNotSatisfiedError` (const std::string &what="")

9.329 PreconditionNotSatisfiedError Class Reference

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

Inheritance diagram for PreconditionNotSatisfiedError:



9.329.1 Detailed Description

Specialized error.

Thrown upon an unsatisfied precondition.

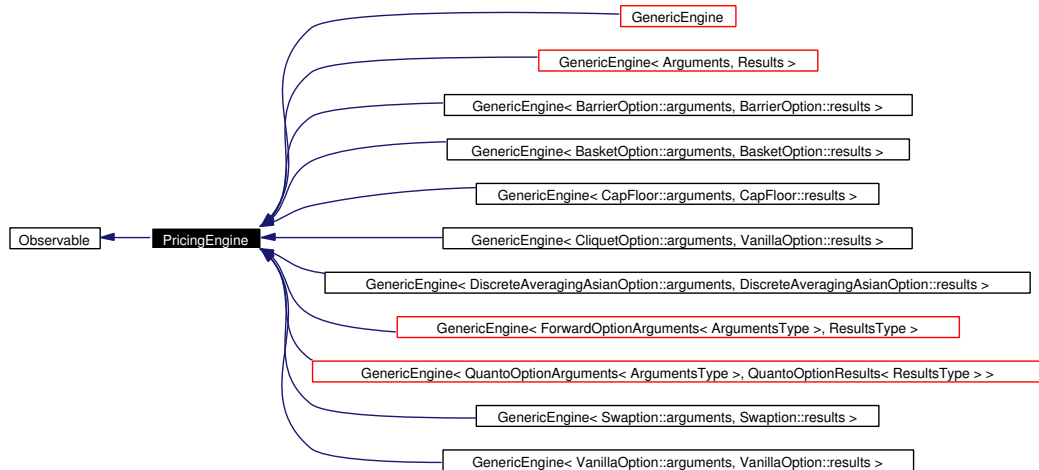
Public Member Functions

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

9.330 PricingEngine Class Reference

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

Inheritance diagram for PricingEngine:



9.330.1 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

9.331 PrimeNumbers Class Reference

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

9.331.1 Detailed Description

Prime numbers calculator.

Taken from "Monte Carlo Methods in Finance", by Peter Jäckel

Static Public Member Functions

- unsigned long **get** (Size absoluteIndex)
Get and store one after another.

9.332 Problem Class Reference

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

9.332.1 Detailed Description

Constrained optimization problem.

Public Member Functions

- **Problem** (**CostFunction** &f, **Constraint** &c, **OptimizationMethod** &meth)
default constructor
- double **value** (const **Array** &x) const
call cost function computation and increment evaluation counter
- void **gradient** (**Array** &grad_f, const **Array** &x) const
call cost function gradient computation and increment
- double **valueAndGradient** (**Array** &grad_f, const **Array** &x) const
call cost function computation and it gradient
- **OptimizationMethod** & **method** () const
Constrained optimization method.
- **Constraint** & **constraint** () const
Constraint(p. 215).
- **CostFunction** & **costFunction** () const
Cost function.
- void **minimize** () const
Minimization.
- **Array** & **minimumValue** () const

Protected Attributes

- **CostFunction** & **costFunction_**
Unconstrained cost function.
- **Constraint** & **constraint_**
Constraint(p. 215).
- **OptimizationMethod** & **method_**
constrained optimization method

9.333 processing_iterator Class Template Reference

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

9.333.1 Detailed Description

template<class Iterator, class UnaryFunction> class QuantLib::processing_iterator< Iterator, UnaryFunction >

Iterator mapping a unary function to an underlying sequence.

This iterator advances an underlying iterator and returns the values obtained by applying a unary function to the values such iterator points to.

This class was implemented based on Christopher Baus and Thomas Becker, *Custom Iterators for the STL*, included in the proceedings of the First Workshop on C++ Template Programming, Erfurt, Germany, 2000 (<http://www.oonumerics.org/tmpw00/>)

Public Types

- typedef UnaryFunction::result_type **value_type**
- typedef const value_type * **pointer**
- typedef const value_type & **reference**

Public Member Functions

- **processing_iterator** (const Iterator &, const UnaryFunction &)

Dereferencing

- reference **operator *** () const
- pointer **operator →** () const

Random access

- value_type **operator[]** (int) const

Increment and decrement

- **processing_iterator** & **operator++** ()
- **processing_iterator** **operator++** (int)
- **processing_iterator** & **operator--** ()
- **processing_iterator** **operator--** (int)
- **processing_iterator** & **operator+=** (difference_type)
- **processing_iterator** & **operator-=** (difference_type)
- **processing_iterator** **operator+** (difference_type)
- **processing_iterator** **operator-** (difference_type)

Difference

- difference_type **operator-** (const **processing_iterator**< Iterator, UnaryFunction > &)

Comparisons

- **bool operator==** (const **processing_iterator**< Iterator, UnaryFunction > &)
- **bool operator!=** (const **processing_iterator**< Iterator, UnaryFunction > &)
- **bool operator<** (const **processing_iterator**< Iterator, UnaryFunction > &)
- **bool operator>** (const **processing_iterator**< Iterator, UnaryFunction > &)
- **bool operator<=** (const **processing_iterator**< Iterator, UnaryFunction > &)
- **bool operator>=** (const **processing_iterator**< Iterator, UnaryFunction > &)

Public Attributes

- `typedef< Iterator >::difference_type` **difference_type**

Related Functions

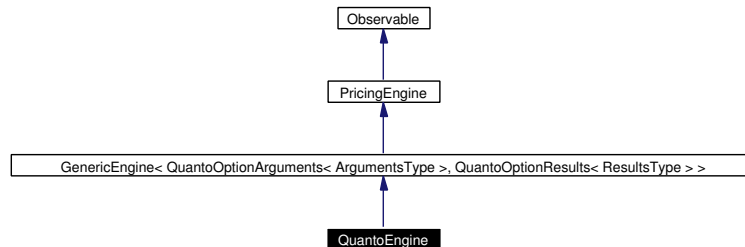
(Note that these are not member functions.)

- **processing_iterator**< Iterator, UnaryFunction > **make_processing_iterator** (Iterator it, UnaryFunction p)
helper function to create processing iterators

9.334 QuantoEngine Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

Inheritance diagram for QuantoEngine:



9.334.1 Detailed Description

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

Quanto engine base class.

Public Member Functions

- **QuantoEngine** (const **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > &)
- void **calculate** () const

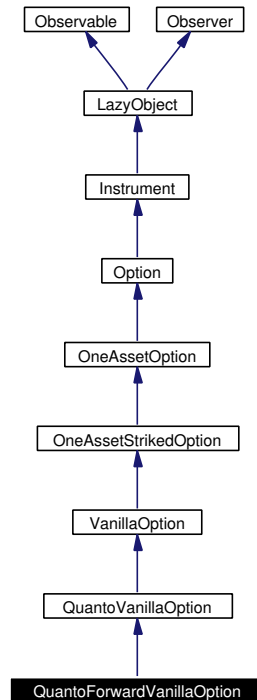
Protected Attributes

- **Handle**< **GenericEngine**< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

9.335 QuantoForwardVanillaOption Class Reference

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

Inheritance diagram for QuantoForwardVanillaOption:



9.335.1 Detailed Description

Quanto version of a forward vanilla option.

Public Types

- typedef **QuantoOptionArguments**< **ForwardVanillaOption::arguments** > **arguments**
- typedef **QuantoOptionResults**< **ForwardVanillaOption::arguments** > **results**

Public Member Functions

- **QuantoForwardVanillaOption** (const **RelinkableHandle**< **TermStructure** > &foreignRiskFreeTS, const **RelinkableHandle**< **BlackVolTermStructure** > &exchRateVolTS, const **RelinkableHandle**< **Quote** > &correlation, double moneyness, **Date** resetDate, const **Handle**< **BlackScholesStochasticProcess** > &stochProc, const **Handle**< **StrikedTypePayoff** > &payoff, const **Handle**< **Exercise** > &exercise, const **Handle**< **PricingEngine** > &engine)
- void **setupArguments** (**Arguments** *) const

9.335.2 Member Function Documentation

9.335.2.1 void setupArguments (Arguments *) const [virtual]

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

Reimplemented from **QuantoVanillaOption** (p. [531](#)).

9.336 QuantoOptionArguments Class Template Reference

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

9.336.1 Detailed Description

```
template<class ArgumentsType> class QuantLib::QuantoOptionArguments< ArgumentsType  
>
```

Arguments for quanto option calculation

Public Member Functions

- void `validate ()` const

Public Attributes

- double `correlation`
- RelinkableHandle< TermStructure > `foreignRiskFreeTS`
- RelinkableHandle< BlackVolTermStructure > `exchRateVolTS`

9.337 QuantoOptionResults Class Template Reference

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

9.337.1 Detailed Description

```
template<class ResultsType> class QuantLib::QuantoOptionResults< ResultsType >
```

Results from quanto option calculation

Public Member Functions

- void `reset ()`

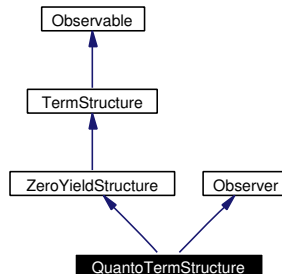
Public Attributes

- double `qvega`
- double `qrho`
- double `qlambda`

9.338 QuantoTermStructure Class Reference

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

Inheritance diagram for QuantoTermStructure:



9.338.1 Detailed Description

Quanto term structure.

Quanto term structure for modelling quanto effect in option pricing.

Note:

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

Public Member Functions

- **QuantoTermStructure** (const **RelinkableHandle**< **TermStructure** > &underlyingDividendTS, const **RelinkableHandle**< **TermStructure** > &riskFreeTS, const **RelinkableHandle**< **TermStructure** > &foreignRiskFreeTS, const **RelinkableHandle**< **BlackVolTermStructure** > &underlyingBlackVolTS, double strike, const **RelinkableHandle**< **BlackVolTermStructure** > &exchRateBlackVolTS, double exchRateATMlevel, double underlyingExchRateCorrelation)

TermStructure interface

- **DayCounter dayCounter** () const
the day counter used for date/time conversion
- **Date todaysDate** () const
today's date
- **Date referenceDate** () const
the reference date, i.e., the date at which discount = 1
- **Date maxDate** () const
the latest date for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- Rate **zeroYieldImpl** (Time, bool extrapolate=false) const
returns the zero yield as seen from the evaluation date

9.338.2 Member Function Documentation

9.338.2.1 void update () [virtual]

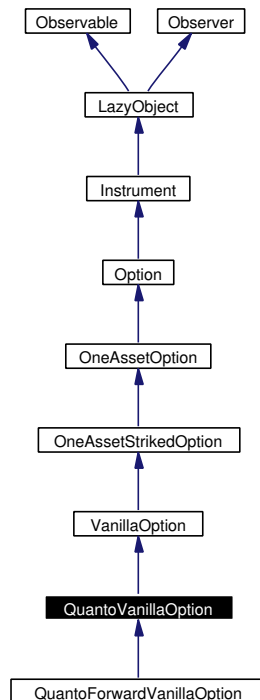
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.339 QuantoVanillaOption Class Reference

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

Inheritance diagram for QuantoVanillaOption:



9.339.1 Detailed Description

quanto version of a vanilla option

Public Types

- typedef **QuantoOptionArguments**< VanillaOption::arguments > **arguments**
- typedef **QuantoOptionResults**< VanillaOption::results > **results**

Public Member Functions

- **QuantoVanillaOption** (const **RelinkableHandle**< **TermStructure** > &foreignRiskFreeTS, const **RelinkableHandle**< **BlackVolTermStructure** > &exchRateVolTS, const **RelinkableHandle**< **Quote** > &correlation, const **Handle**< **BlackScholesStochasticProcess** > &stochProc, const **Handle**< **StrikedTypePayoff** > &payoff, const **Handle**< **Exercise** > &exercise, const **Handle**< **PricingEngine** > &engine)
- void **setupArguments** (**Arguments** *) const

greeks

- double **qvega** () const
- double **qrho** () const
- double **qlambda** () const

Protected Member Functions

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

Protected Attributes

- RelinkableHandle< TermStructure > **foreignRiskFreeTS_**
- RelinkableHandle< BlackVolTermStructure > **exchRateVolTS_**
- RelinkableHandle< Quote > **correlation_**
- double **qvega_**
- double **qrho_**
- double **qlambda_**

9.339.2 Member Function Documentation

9.339.2.1 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** (p. 482).

Reimplemented in **QuantoForwardVanillaOption** (p. 525).

9.339.2.2 void setupExpired () const [protected, virtual]

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

Reimplemented from **OneAssetOption** (p. 478).

9.339.2.3 void performCalculations () const [protected, virtual]

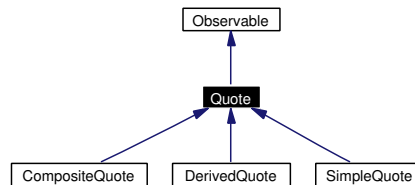
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **VanillaOption** (p. 624).

9.340 Quote Class Reference

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

Inheritance diagram for Quote:



9.340.1 Detailed Description

purely virtual base class for market observables

Public Member Functions

- virtual double **value** () const=0
returns the current value

9.341 RandomArrayGenerator Class Template Reference

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

9.341.1 Detailed Description

```
template<class RNG> class QuantLib::RandomArrayGenerator< RNG >
```

Generates random arrays using a random number generator.

Deprecated

use `RandomSequenceGenerator`(p. [534](#)) instead.

Public Types

- `typedef Sample< Array > sample_type`

Public Member Functions

- `RandomArrayGenerator` (const `Array` &variance, long seed=0)
- `RandomArrayGenerator` (const `Matrix` &covariance, long seed=0)
- const `sample_type` & `next` () const
- int `size` () const

9.342 RandomSequenceGenerator Class Template Reference

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

9.342.1 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, long seed=0)
- const sample_type & **nextSequence** () const
- std::vector< unsigned long > **nextInt32Sequence** () const
- const sample_type & **lastSequence** () const
- Size **dimension** () const

9.343 RateFormatter Class Reference

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

9.343.1 Detailed Description

Formats rates for output.

Formatting is in percentage form (xx.xxxxx)

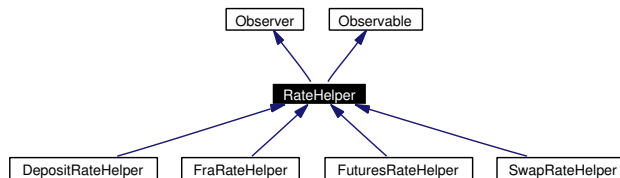
Static Public Member Functions

- `std::string toString` (double rate, int precision=5)

9.344 RateHelper Class Reference

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

Inheritance diagram for RateHelper:



9.344.1 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**(p. 583) contains a **Swap**(p. 581) instrument for the time being.

Public Member Functions

- **RateHelper** (const **RelinkableHandle**< **Quote** > "e)
- **RateHelper** (double quote)

RateHelper interface

- double **quoteError** () const
- double **referenceQuote** () const
- virtual double **impliedQuote** () const=0
- virtual **DiscountFactor** **discountGuess** () const
- virtual void **setTermStructure** (**TermStructure** *)
sets the term structure to be used for pricing
- virtual **Date** **maturity** () const=0
maturity date

Observer interface

- void **update** ()

Protected Attributes

- **RelinkableHandle**< **Quote** > quote_
- **TermStructure** * termStructure_

9.344.2 Member Function Documentation

9.344.2.1 virtual void setTermStructure (TermStructure *) [virtual]

sets the term structure to be used for pricing

Warning:

Being a pointer and not a **Handle**(p. 339), the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented in **DepositRateHelper** (p. 243), **FraRateHelper** (p. 318), and **SwapRateHelper** (p. 584).

9.344.2.2 void update () [virtual]

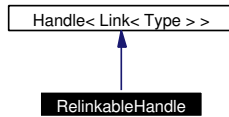
This method must be implemented in derived classes. An instance of **Observer** does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. 475).

9.345 RelinkableHandle Class Template Reference

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

Inheritance diagram for RelinkableHandle:



9.345.1 Detailed Description

```
template<class Type> class QuantLib::RelinkableHandle< Type >
```

Globally accessible relinkable pointer.

An instance of this class can be relinked to another **Handle**(p. 339): such change will be propagated to all the copies of the instance.

Precondition:

Class "Type" must inherit from **Observable**(p. 471)

Public Member Functions

- **RelinkableHandle** (const **Handle**< Type > &h=**Handle**< Type >(), bool registerAsObserver=true)
- void **linkTo** (const **Handle**< Type > &h, bool registerAsObserver=true)
- const **Handle**< Type > & **operator** → () const
dereferencing
- bool **isNull** () const
Checks if the contained handle points to anything.

9.345.2 Constructor & Destructor Documentation

9.345.2.1 **RelinkableHandle** (const **Handle**< Type > & h = **Handle**< Type >(), bool *registerAsObserver* = true) [explicit]

Warning:

see the documentation of **Link**(p. 404) for issues relatives to **registerAsObserver**.

9.345.3 Member Function Documentation

9.345.3.1 void **linkTo** (const **Handle**< Type > & h, bool *registerAsObserver* = true)

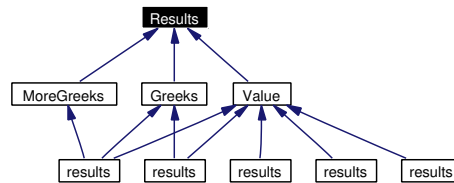
Warning:

see the documentation of **Link**(p. 404) for issues relatives to **registerAsObserver**.

9.346 Results Class Reference

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

Inheritance diagram for Results:



9.346.1 Detailed Description

base class for generic result groups

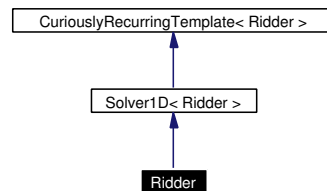
Public Member Functions

- virtual void **reset** ()=0

9.347 Ridder Class Reference

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

Inheritance diagram for Ridder:



9.347.1 Detailed Description

Ridder 1-D solver

Public Member Functions

- `template<class F> double solveImpl (const F &f, double xAcc) const`

9.348 SalvagingAlgorithm Struct Reference

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

9.348.1 Detailed Description

algorithm used for matricial pseudo square root

Public Types

- enum Type { None, Spectral, Hypersphere }

9.349 Sample Struct Template Reference

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

9.349.1 Detailed Description

```
template<class T> struct QuantLib::Sample< T >
```

weighted sample

Public Types

- `typedef T value_type`

Public Member Functions

- `Sample (const T &value, double weight)`

Public Attributes

- `T value`
- `double weight`

9.350 Schedule Class Reference

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

9.350.1 Detailed Description

Payment schedule.

Iterators

- typedef std::vector< **Date** >::const_iterator **const_iterator**
- const_iterator **begin** () const
- const_iterator **end** () const

Public Member Functions

- **Schedule** (const **Calendar** &calendar, const **Date** &startDate, const **Date** &endDate, int frequency, RollingConvention rollingConvention, bool isAdjusted, const **Date** &stubDate=**Date**(), bool startFromEnd=false, bool longFinal=false)
- **Schedule** (const std::vector< **Date** > &, const **Calendar** &calendar, RollingConvention rollingConvention, bool isAdjusted)

Date access

- Size **size** () const
- const **Date** & **operator[]** (Size i) const
- const **Date** & **date** (Size i) const
- bool **isRegular** (Size i) const

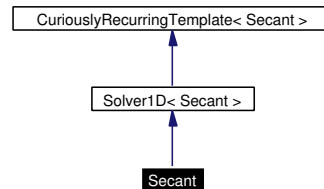
Other inspectors

- const **Calendar** & **calendar** () const
- const **Date** & **startDate** () const
- const **Date** & **endDate** () const
- Frequency **frequency** () const
- RollingConvention **rollingConvention** () const
- bool **isAdjusted** () const

9.351 Secant Class Reference

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

Inheritance diagram for Secant:



9.351.1 Detailed Description

Secant 1-D solver

Public Member Functions

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

9.352 SegmentIntegral Class Reference

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

9.352.1 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 $\Delta x = (b - a)/N$.

Public Member Functions

- **SegmentIntegral** (Size intervals)
- `template<class F> double operator() (const F &f, double a, double b) const`

9.353 SequenceStatistics Class Template Reference

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

9.353.1 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.

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](#)(p. 418)
- **Disposable< Matrix > correlation** () const
returns the correlation [Matrix](#)(p. 418)

1-D inspectors lifted from underlying statistics class

- Size **samples** () const
- double **weightSum** () const

N-D inspectors lifted from underlying statistics class

- std::vector< double > **mean** () const
- std::vector< double > **variance** () const
- std::vector< double > **standardDeviation** () const
- std::vector< double > **downsideVariance** () const

- `std::vector< double > downsideDeviation () const`
- `std::vector< double > semiVariance () const`
- `std::vector< double > semiDeviation () const`
- `std::vector< double > errorEstimate () const`
- `std::vector< double > skewness () const`
- `std::vector< double > kurtosis () const`
- `std::vector< double > min () const`
- `std::vector< double > max () const`
- `std::vector< double > gaussianPercentile (double y) const`
- `std::vector< double > percentile (double y) const`
- `std::vector< double > gaussianPotentialUpside (double percentile) const`
- `std::vector< double > potentialUpside (double percentile) const`
- `std::vector< double > gaussianValueAtRisk (double percentile) const`
- `std::vector< double > valueAtRisk (double percentile) const`
- `std::vector< double > gaussianExpectedShortfall (double percentile) const`
- `std::vector< double > expectedShortfall (double percentile) const`
- `std::vector< double > regret (double target) const`
- `std::vector< double > gaussianShortfall (double target) const`
- `std::vector< double > shortfall (double target) const`
- `std::vector< double > gaussianAverageShortfall (double target) const`
- `std::vector< double > averageShortfall (double target) const`

Modifiers

- `void reset (Size dimension=0)`
- `template<class Sequence> void add (const Sequence &sample, double weight=1.0)`
- `template<class Iterator> void add (Iterator begin, Iterator end, double weight=1.0)`

Protected Attributes

- Size `dimension_`
- `std::vector< statistics_type > stats_`
- `std::vector< double > results_`
- Matrix `quadraticSum_`

9.354 Short Class Template Reference

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

9.354.1 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** (double nominal, const **Date** &paymentDate, const **Handle**< **Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, Spread spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())
- double **amount** () const
inhibit calculation

9.354.2 Member Function Documentation

9.354.2.1 double amount () const

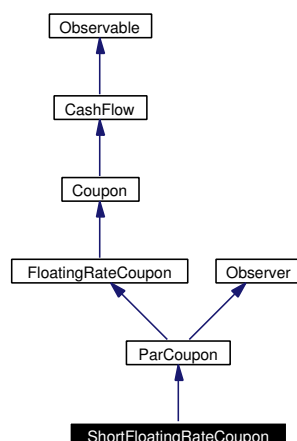
inhibit calculation

Unlike **ParCoupon**(p. 499), this coupon can't calculate its fixing for future dates, either.

9.355 ShortFloatingRateCoupon Class Reference

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

Inheritance diagram for ShortFloatingRateCoupon:



9.355.1 Detailed Description

Short coupon at par on a term structure

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **ShortFloatingRateCoupon** (double nominal, const **Date** &paymentDate, const **Handle**<**Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, Spread spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**())
- double **amount** () const
throws when an interpolated fixing is needed

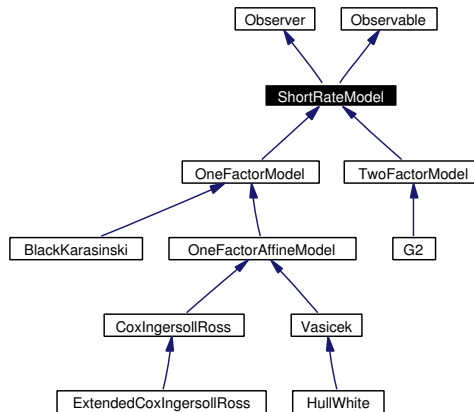
Visitability

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

9.356 ShortRateModel Class Reference

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

Inheritance diagram for ShortRateModel:



9.356.1 Detailed Description

Abstract short-rate model class.

Public Member Functions

- **ShortRateModel** (Size nArguments)
- void **update** ()
- virtual **Handle**< **Lattice** > **tree** (const **TimeGrid** &grid) const=0
- void **calibrate** (const std::vector< **Handle**< **CalibrationHelper** > > &instruments, **OptimizationMethod** &method, const **Constraint** &constraint=**Constraint**())
Calibrate to a set of market instruments (caps/swaptions).
- const **Handle**< **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_**
- **Handle**< **Constraint** > **constraint_**

Friends

- class `CalibrationFunction`

9.356.2 Member Function Documentation

9.356.2.1 `void update ()` [virtual]

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

Implements `Observer` (p. [475](#)).

9.356.2.2 `void calibrate (const std::vector< Handle< CalibrationHelper > > & instruments, OptimizationMethod & method, const Constraint & constraint = Constraint())`

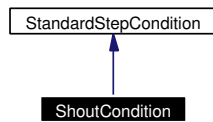
Calibrate to a set of market instruments (caps/swaptions).

An additional constraint can be passed which must be satisfied in addition to the constraints of the model.

9.357 ShoutCondition Class Reference

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

Inheritance diagram for ShoutCondition:



9.357.1 Detailed Description

Shout option condition.

A shout option is an option where the holder has the right to lock in a minimum value for the payoff at one (shout) time during the option's life. The minimum value is the option's intrinsic value at the shout time.

Todo

Unify the intrinsicValues/Payoff thing

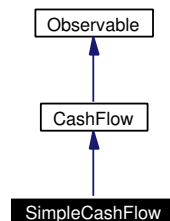
Public Member Functions

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

9.358 SimpleCashFlow Class Reference

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

Inheritance diagram for SimpleCashFlow:



9.358.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

Public Member Functions

- **SimpleCashFlow** (double amount, const **Date** &date)

CashFlow interface

- double **amount** () const
returns the amount of the cash flow
- **Date** **date** () const
returns the date at which the cash flow is settled

Visitability

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

9.358.2 Member Function Documentation

9.358.2.1 double amount () const [virtual]

returns the amount of the cash flow

Note:

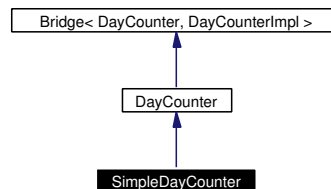
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements **CashFlow** (p. [200](#)).

9.359 SimpleDayCounter Class Reference

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

Inheritance diagram for SimpleDayCounter:



9.359.1 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.

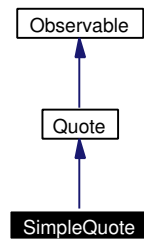
Warning:

this day counter should be used together with **NullCalendar**([p. 468](#)), which ensures that dates at whole-month distances share the same day of month. It is **not** guaranteed to work with any other calendar.

9.360 SimpleQuote Class Reference

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

Inheritance diagram for SimpleQuote:



9.360.1 Detailed Description

market element returning a stored value

Public Member Functions

- **SimpleQuote** (double value)

Quote interface

- double **value** () const
returns the current value

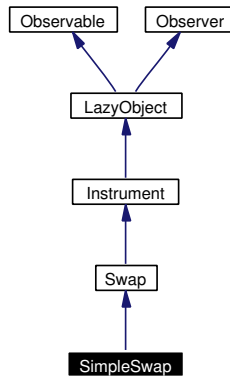
Modifiers

- void **setValue** (double value)

9.361 SimpleSwap Class Reference

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

Inheritance diagram for SimpleSwap:



9.361.1 Detailed Description

Simple fixed-rate vs Libor swap.

Public Member Functions

- **SimpleSwap** (bool payFixedRate, const **Date** &startDate, int n, TimeUnit units, const **Calendar** &calendar, RollingConvention rollingConvention, double nominal, int fixedFrequency, Rate fixedRate, bool fixedIsAdjusted, const **DayCounter** &fixedDayCount, int floatingFrequency, const **Handle**< **Xibor** > &index, int indexFixingDays, Spread spread, const **RelinkableHandle**< **TermStructure** > &termStructure)
- **SimpleSwap** (bool payFixedRate, double nominal, const **Schedule** &fixedSchedule, Rate fixedRate, const **DayCounter** &fixedDayCount, const **Schedule** &floatSchedule, const **Handle**< **Xibor** > &index, int indexFixingDays, Spread spread, const **RelinkableHandle**< **TermStructure** > &termStructure)
- Rate **fairRate** () const
- Spread **fairSpread** () const
- double **fixedLegBPS** () const
- double **floatingLegBPS** () const
- Rate **fixedRate** () const
- Spread **spread** () const
- double **nominal** () const
- bool **payFixedRate** () const
- const std::vector< **Handle**< **CashFlow** > > & **fixedLeg** () const
- const std::vector< **Handle**< **CashFlow** > > & **floatingLeg** () const
- void **setupArguments** (**Arguments** *args) const

9.361.2 Member Function Documentation

9.361.2.1 void setupArguments (Arguments * *args*) const [virtual]

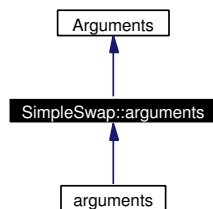
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from **Instrument** (p. [368](#)).

9.362 SimpleSwap::arguments Class Reference

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

Inheritance diagram for SimpleSwap::arguments:



9.362.1 Detailed Description

Arguments for simple swap calculation

Public Member Functions

- void **validate** () const

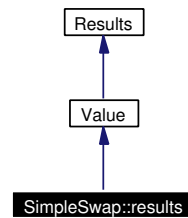
Public Attributes

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

9.363 SimpleSwap::results Class Reference

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

Inheritance diagram for SimpleSwap::results:



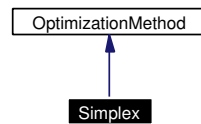
9.363.1 Detailed Description

Results from simple swap calculation

9.364 Simplex Class Reference

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

Inheritance diagram for Simplex:



9.364.1 Detailed Description

Multi-dimensional simplex class.

Public Member Functions

- **Simplex** (double *lambda*, double *tol*)
- virtual void **minimize** (const **Problem** &P) const
minimize the optimization problem P

9.364.2 Constructor & Destructor Documentation

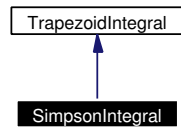
9.364.2.1 Simplex (double *lambda*, double *tol*)

Constructor taking as input the characteristic length and tolerance

9.365 SimpsonIntegral Class Reference

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

Inheritance diagram for SimpsonIntegral:



9.365.1 Detailed Description

Integral of a one-dimensional function.

Public Member Functions

- **SimpsonIntegral** (double accuracy, Size maxIterations=**Null**< int >())
- template<class F> double **operator()** (const F &f, double a, double b) const

- double `theta_`
- bool `rhoComputed_`
- bool `dividendRhoComputed_`
- bool `vegaComputed_`
- bool `thetaComputed_`

Static Protected Attributes

- const double `dVolMultiplier_`
- const double `dRMultiplier_`

Friends

- class `VolatilityFunction`
- class `DivYieldFunction`

9.366.2 Member Function Documentation

9.366.2.1 `double impliedVolatility (double targetValue, double accuracy = 1e-4, Size maxEvaluations = 100, double minVol = QL_MIN_VOLATILITY, double maxVol = QL_MAX_VOLATILITY) const`

Warning:

Options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g. binary options. In these cases `impliedVolatility` can fail and in any case is meaningless. Another possible source of failure is to have a `targetValue` that is not attainable with any volatility, e.g. a `targetValue` lower than the intrinsic value in the case of American options.

9.367 SobolRsg Class Reference

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

9.367.1 Detailed Description

Sobol low-discrepancy sequence generator.

A Gray code counter and bitwise operations are used for very fast sequence generation.

The implementation relies on primitive polynomials modulo two and initialization numbers from the book "Monte Carlo Methods in Finance" by Peter Jäckel.

21200 primitive polynomials modulo two are provided by default. There are 8 129 334 polynomials as provided by Jäckel which can be downloaded from quantlib.org. If you need that many dimensions you must replace the `primitivepolynomial.*` files with the ones downloaded and recompile the library.

The choice of initialization numbers is crucial for the homogeneity properties of the sequence. Jäckel's initialization numbers are superior to the "unit initialization" suggested in "Numerical Recipes in C" by Press, Teukolsky, Vetterling, and Flannery.

For more info on Sobol sequences see "Monte Carlo Methods in Finance", by Peter Jäckel, section 8.3 and "Numerical Recipes in C", 2nd edition, by Press, Teukolsky, Vetterling, and Flannery, section 7.7.

Public Types

- `typedef Sample< Array > sample_type`

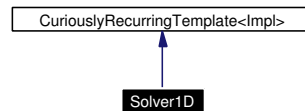
Public Member Functions

- **SobolRsg** (Size dimensionality, unsigned long seed=0, bool unitInitialization=false)
- `const sample_type & nextSequence () const`
- `const sample_type & lastSequence () const`
- Size **dimension** () const

9.368 Solver1D Class Template Reference

```
#include <ql/solver1d.hpp>
```

Inheritance diagram for Solver1D:



9.368.1 Detailed Description

```
template<class Impl> class QuantLib::Solver1D< Impl >
```

Base class for 1-D solvers.

The implementation of this class uses the so-called "Barton-Nackman trick", also known as "the curiously recurring template pattern". Concrete solvers will be declared as:

```

class Foo : public Solver1D<Foo> {
public:
    ...
    template <class F>
    double solveImpl(const F& f, double accuracy) const {
        ...
    }
};

```

Before calling `solveImpl`, the base class will set its protected data members so that:

- `xMin_` and `xMax_` form a valid bracket;
- `fxMin_` and `fxMax_` contain the values of the function in `xMin_` and `xMax_`;
- `root_` is a valid initial guess. The implementation of `solveImpl` can safely assume all of the above.

Todo

- a) Clean up the interface so that it is clear whether the accuracy is specified for `x` or `f(x)`.
- b) Add target value (now the target value is 0.0)

Public Member Functions

Modifiers

- `template<class F> double solve (const F &f, double accuracy, double guess, double step) const`
- `template<class F> double solve (const F &f, double accuracy, double guess, double xMin, double xMax) const`
- `void setMaxEvaluations (Size evaluations)`
- `void setLowerBound (double lowerBound)`
sets the lower bound for the function domain

- void **setUpperBound** (double upperBound)
sets the upper bound for the function domain

Protected Attributes

- double **root_**
- double **xMin_**
- double **xMax_**
- double **fxMin_**
- double **fxMax_**
- Size **maxEvaluations_**
- Size **evaluationNumber_**

9.368.2 Member Function Documentation

9.368.2.1 double solve (const F & *f*, double *accuracy*, double *guess*, double *step*) const

This method returns the zero of the function f , determined with the given accuracy (i.e., x is considered a zero if $|f(x)| < accuracy$). This method contains a bracketing routine to which an initial guess must be supplied as well as a step used to scan the range of the possible bracketing values.

9.368.2.2 double solve (const F & *f*, double *accuracy*, double *guess*, double *xMin*, double *xMax*) const

This method returns the zero of the function f , determined with the given accuracy (i.e., x is considered a zero if $|f(x)| < accuracy$). An initial guess must be supplied, as well as two values x_{\min} and x_{\max} which must bracket the zero (i.e., either $f(x_{\min}) \leq 0 \leq f(x_{\max})$, or $f(x_{\max}) \leq 0 \leq f(x_{\min})$ must be true).

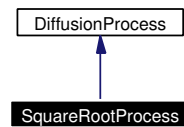
9.368.2.3 void setMaxEvaluations (Size *evaluations*)

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.

9.369 SquareRootProcess Class Reference

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

Inheritance diagram for SquareRootProcess:



9.369.1 Detailed Description

Square-root process class.

This class describes a square-root process governed by

$$dx = a(b - x_t)dt + \sigma \sqrt{x_t}dW_t.$$

Public Member Functions

- **SquareRootProcess** (double b, double a, double sigma, double x0=0)
- double **drift** (Time t, double x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- double **diffusion** (Time t, double x) const

9.369.2 Member Function Documentation

9.369.2.1 double diffusion (Time t, double x) const [virtual]

returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

Implements **DiffusionProcess** (p. [246](#)).

9.370 StatsHolder Class Reference

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

9.370.1 Detailed Description

Helper class for precomputed distributions.

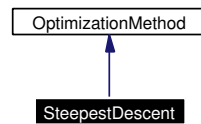
Public Member Functions

- **StatsHolder** (double mean, double standardDeviation)
- double **mean** () const
- double **standardDeviation** () const

9.371 SteepestDescent Class Reference

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

Inheritance diagram for SteepestDescent:



9.371.1 Detailed Description

Multi-dimensional steepest-descent class.

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

search direction = $-f'(x)$

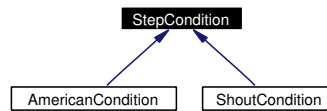
Public Member Functions

- **SteepestDescent** ()
default default constructor (msvc bug)
- **SteepestDescent** (const **Handle**< **LineSearch** > &lineSearch)
default constructor
- virtual ~**SteepestDescent** ()
destructor
- virtual void **minimize** (const **Problem** &P) const
minimize the optimization problem P

9.372 StepCondition Class Template Reference

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

Inheritance diagram for StepCondition:



9.372.1 Detailed Description

```
template<class arrayType> class QuantLib::StepCondition< arrayType >
```

condition to be applied at every time step

Public Member Functions

- virtual void **applyTo** (arrayType &a, Time t) const=0
- virtual void **applyTo** (Handle< DiscretizedAsset > asset) const=0

9.373 stepping_iterator Class Template Reference

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

9.373.1 Detailed Description

template<class RandomAccessIterator> class QuantLib::stepping_iterator< RandomAccessIterator >

Iterator advancing in constant steps.

This iterator advances an underlying random access iterator in steps of n positions, where n is an integer given upon construction.

Public Member Functions

- **stepping_iterator** (const RandomAccessIterator &it, difference_type step)

Dereferencing

- reference **operator *** () const
- pointer **operator →** () const

Random access

- reference **operator []** (int i) const

Increment and decrement

- **stepping_iterator & operator++** ()
- **stepping_iterator operator++** (int)
- **stepping_iterator & operator--** ()
- **stepping_iterator operator--** (int)
- **stepping_iterator & operator+=** (difference_type i)
- **stepping_iterator & operator-=** (difference_type i)
- **stepping_iterator operator+** (difference_type i)
- **stepping_iterator operator-** (difference_type i)

Difference

- difference_type **operator-** (const **stepping_iterator** &i)

Comparisons

- bool **operator==** (const **stepping_iterator** &i)
- bool **operator!=** (const **stepping_iterator** &i)
- bool **operator<** (const **stepping_iterator** &i)
- bool **operator>** (const **stepping_iterator** &i)
- bool **operator<=** (const **stepping_iterator** &i)
- bool **operator>=** (const **stepping_iterator** &i)

Public Attributes

- `typedef< RandomAccessIterator >::difference_type` **difference_type**
- `typedef< RandomAccessIterator >::pointer` **pointer**
- `typedef< RandomAccessIterator >::reference` **reference**

Related Functions

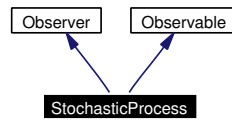
(Note that these are not member functions.)

- **stepping_iterator< Iterator > make_stepping_iterator** (Iterator it, typename **stepping_iterator< Iterator >::difference_type** step)
helper function to create stepping iterators

9.374 StochasticProcess Class Reference

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

Inheritance diagram for StochasticProcess:



9.374.1 Detailed Description

Base stochastic process class.

Just an arguments placeholder for the time being. To be merged/refactored with **Diffusion-Process**(p. [246](#))

Public Member Functions

- void **update** ()

9.374.2 Member Function Documentation

9.374.2.1 void update () [virtual]

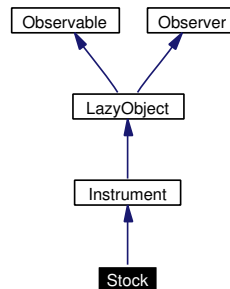
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer** (p. [475](#)).

9.375 Stock Class Reference

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

Inheritance diagram for Stock:



9.375.1 Detailed Description

Simple stock class.

Public Member Functions

- **Stock** (const **RelinkableHandle**< **Quote** > "e)
- **bool isExpired** () const
returns whether the instrument is still tradable.

Protected Member Functions

- **void performCalculations** () const

9.375.2 Member Function Documentation

9.375.2.1 **void performCalculations** () const [protected, virtual]

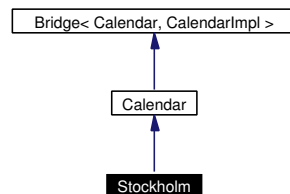
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **Instrument** (p. [369](#)).

9.376 Stockholm Class Reference

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

Inheritance diagram for Stockholm:



9.376.1 Detailed Description

Stockholm calendar

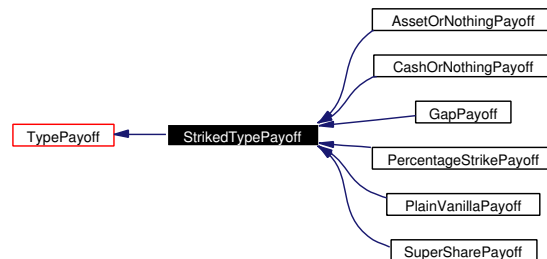
Holidays:

- 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

9.377 StrikedTypePayoff Class Reference

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

Inheritance diagram for StrikedTypePayoff:



9.377.1 Detailed Description

Intermediate class for payoffs based on a fixed strike.

Public Member Functions

- **StrikedTypePayoff** (Option::Type type, double strike)
- double **strike** () const
- void **setStrike** (double strike)

Protected Attributes

- double **strike_**

9.378 StringFormatter Class Reference

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

9.378.1 Detailed Description

Formats strings as lower- or uppercase.

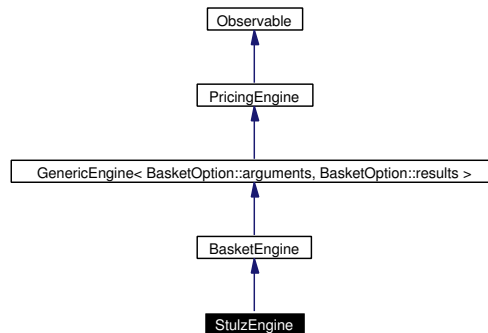
Static Public Member Functions

- `std::string toLowercase` (const std::string &s)
- `std::string toUppercase` (const std::string &s)

9.379 StulzEngine Class Reference

```
#include <ql/PricingEngines/Basket/stulzengine.hpp>
```

Inheritance diagram for StulzEngine:



9.379.1 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.

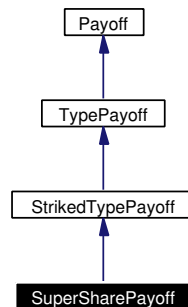
Public Member Functions

- void **calculate** () const

9.380 SuperSharePayoff Class Reference

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

Inheritance diagram for SuperSharePayoff:



9.380.1 Detailed Description

Binary supershare payoff.

Public Member Functions

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

9.381 SVD Class Reference

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

9.381.1 Detailed Description

Singular value decomposition.

Refer to Golub and Van Loan: **Matrix**(p. [418](#)) computation, The Johns Hopkins University Press

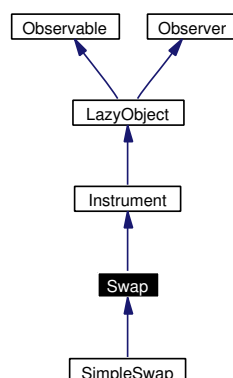
Public Member Functions

- **SVD** (const **Matrix** &)
- const **Matrix** & **U** () const
- const **Matrix** & **V** () const
- const **Array** & **singularValues** () const
- **Disposable**< **Matrix** > **S** () const
- double **norm2** ()
- double **cond** ()
- int **rank** ()

9.382 Swap Class Reference

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

Inheritance diagram for Swap:



9.382.1 Detailed Description

Interest rate swap.

The cash flows belonging to the first leg are paid; the ones belonging to the first leg are received.

Public Member Functions

- **Swap** (const std::vector< **Handle**< **CashFlow** > > &firstLeg, const std::vector< **Handle**< **CashFlow** > > &secondLeg, const **RelinkableHandle**< **TermStructure** > &termStructure)

Instrument interface

- **bool isExpired () const**
returns whether the instrument is still tradable.

Additional interface

- **Date startDate () const**
- **Date maturity () const**
- **double firstLegBPS () const**
- **double secondLegBPS () const**
- **TimeBasket sensitivity (int basis=2) const**

Protected Member Functions

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

Protected Attributes

- `std::vector< Handle< CashFlow > > firstLeg_`
- `std::vector< Handle< CashFlow > > secondLeg_`
- `RelinkableHandle< TermStructure > termStructure_`
- `double firstLegBPS_`
- `double secondLegBPS_`

9.382.2 Member Function Documentation

9.382.2.1 TimeBasket sensitivity (int *basis* = 2) const

Bug

This method must still be checked. It is not guaranteed to yield the right results.

9.382.2.2 void setupExpired () const [protected, virtual]

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

Reimplemented from **Instrument** (p. [369](#)).

9.382.2.3 void performCalculations () const [protected, virtual]

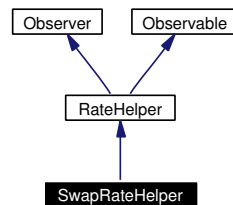
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **Instrument** (p. [369](#)).

9.383 SwapRateHelper Class Reference

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

Inheritance diagram for SwapRateHelper:



9.383.1 Detailed Description

Swap rate

Warning:

This class assumes that the settlement date does not change between calls of **setTermStructure()**(p. 584).

Public Member Functions

- **SwapRateHelper** (const **RelinkableHandle**< **Quote** > &rate, int n, TimeUnit units, int settlementDays, const **Calendar** &calendar, RollingConvention convention, int fixedFrequency, bool fixedIsAdjusted, const **DayCounter** &fixedDayCount, int floatingFrequency)
- **SwapRateHelper** (double rate, int n, TimeUnit units, int settlementDays, const **Calendar** &calendar, RollingConvention convention, int fixedFrequency, bool fixedIsAdjusted, const **DayCounter** &fixedDayCount, int floatingFrequency)
- double **impliedQuote** () const
- **Date maturity** () const
maturity date
- void **setTermStructure** (**TermStructure** *)
sets the term structure to be used for pricing

Protected Attributes

- int **n_**
- TimeUnit **units_**
- int **settlementDays_**
- **Calendar** **calendar_**
- RollingConvention **convention_**
- int **fixedFrequency_**
- int **floatingFrequency_**
- bool **fixedIsAdjusted_**

- `DayCounter fixedDayCount_`
- `Date settlement_`
- `Handle< SimpleSwap > swap_`
- `RelinkableHandle< TermStructure > termStructureHandle_`

9.383.2 Member Function Documentation

9.383.2.1 `void setTermStructure (TermStructure *)` [virtual]

sets the term structure to be used for pricing

Warning:

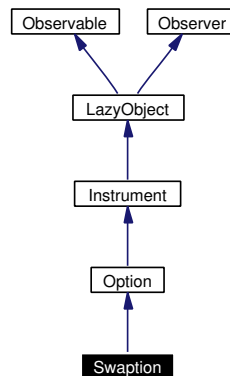
Being a pointer and not a **Handle**(p. 339), the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from **RateHelper** (p. 537).

9.384 Swaption Class Reference

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

Inheritance diagram for Swaption:



9.384.1 Detailed Description

Swaption class

Public Member Functions

- **Swaption** (const **Handle**< **SimpleSwap** > &swap, const **Handle**< **Exercise** > &exercise, const **RelinkableHandle**< **TermStructure** > &termStructure, const **Handle**< **PricingEngine** > &engine)
- **bool isExpired** () const
returns whether the instrument is still tradable.
- **void setupArguments** (**Arguments** *) const

Protected Member Functions

- **void performCalculations** () const

9.384.2 Member Function Documentation

9.384.2.1 **void setupArguments** (**Arguments** *) const [virtual]

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

Reimplemented from **Instrument** (p. 368).

9.384.2.2 void performCalculations () const [protected, virtual]

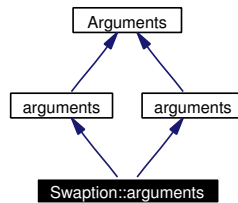
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from **Instrument** (p. [369](#)).

9.385 Swaption::arguments Class Reference

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

Inheritance diagram for Swaption::arguments:



9.385.1 Detailed Description

Arguments for swaption calculation

Public Member Functions

- void **validate** () const

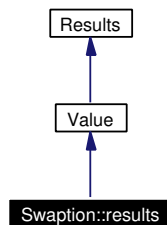
Public Attributes

- Rate **fairRate**
- Rate **fixedRate**
- double **fixedBPS**

9.386 Swaption::results Class Reference

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

Inheritance diagram for Swaption::results:



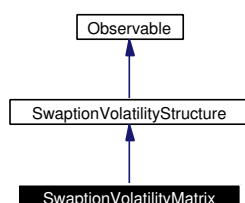
9.386.1 Detailed Description

Results from swaption calculation

9.387 SwaptionVolatilityMatrix Class Reference

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

Inheritance diagram for SwaptionVolatilityMatrix:



9.387.1 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.

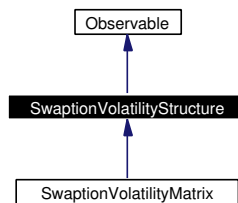
Public Member Functions

- **SwaptionVolatilityMatrix** (const **Date** &todayDate, const std::vector< **Date** > &exerciseDates, const std::vector< **Period** > &lengths, const **Matrix** &volatilities, const **DayCounter** &dayCounter=**Thirty360**())
- **Date** todayDate () const
returns today's date
- **DayCounter** dayCounter () const
returns the day counter used for internal date/time conversions
- const std::vector< **Date** > & exerciseDates () const
- const std::vector< **Period** > & lengths () const

9.388 SwaptionVolatilityStructure Class Reference

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

Inheritance diagram for SwaptionVolatilityStructure:



9.388.1 Detailed Description

Swaption-volatility structure

This class is purely abstract and defines the interface of concrete swaption volatility structures which will be derived from this one.

Public Member Functions

- virtual **Date** **todayDate** () const=0
returns today's date
- virtual **DayCounter** **dayCounter** () const=0
returns the day counter used for internal date/time conversions
- double **volatility** (const **Date** &start, const **Period** &length, Rate strike) const
returns the volatility for a given starting date and length
- double **volatility** (Time start, Time length, Rate strike) const
returns the volatility for a given starting time and length

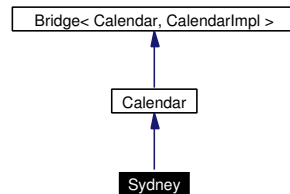
Protected Member Functions

- virtual double **volatilityImpl** (Time start, Time length, Rate strike) const=0
implements the actual volatility calculation in derived classes
- virtual std::pair< Time, Time > **convertDates** (const **Date** &start, const **Period** &length) const
implements the conversion between dates and times

9.389 Sydney Class Reference

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

Inheritance diagram for Sydney:



9.389.1 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)

9.390 SymmetricSchurDecomposition Class Reference

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

9.390.1 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

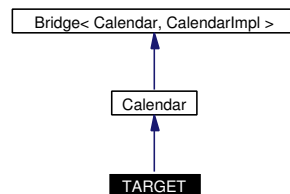
Public Member Functions

- **SymmetricSchurDecomposition** (const **Matrix** &s)
- const **Array** & **eigenvalues** () const
- const **Matrix** & **eigenvectors** () const

9.391 TARGET Class Reference

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

Inheritance diagram for TARGET:



9.391.1 Detailed Description

TARGET calendar

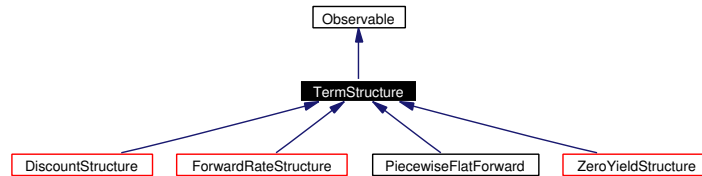
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday (since 2000)
- Easter Monday (since 2000)
- Labour Day, May 1st (since 2000)
- Christmas, December 25th
- Day of Goodwill, December 26th (since 2000)
- December 31st (1998, 1999, and 2001)

9.392 TermStructure Class Reference

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

Inheritance diagram for TermStructure:



9.392.1 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, **DiscountStructure**(p. 251), **ForwardRateStructure**(p. 311)
- allow for different compounding rules and compounding frequencies

Public Member Functions

Rates and discount

These methods are either function of dates or times. In the latter case, times are calculated as fraction of year from the reference date.

- Rate **zeroYield** (const **Date** &, bool extrapolate=false) const
zero-yield rate
- Rate **zeroYield** (Time, bool extrapolate=false) const
zero-yield rate
- DiscountFactor **discount** (const **Date** &, bool extrapolate=false) const
discount factor
- DiscountFactor **discount** (Time, bool extrapolate=false) const
discount factor
- Rate **instantaneousForward** (const **Date** &, bool extrapolate=false) const
instantaneous forward rate
- Rate **instantaneousForward** (Time, bool extrapolate=false) const
instantaneous forward rate

- Rate **compoundForward** (const **Date** &, int, bool extrapolate=false) const
instantaneous forward rate at a given compounding frequency
- Rate **compoundForward** (Time, int, bool extrapolate=false) const
instantaneous forward rate at a given compounding frequency
- Rate **forward** (const **Date** &, const **Date** &, bool extrapolate=false) const
discrete forward rate between two dates
- Rate **forward** (Time, Time, bool extrapolate=false) const
discrete forward rate between two times
- Rate **zeroCoupon** (const **Date** &, int, bool extrapolate=false) const
zero-coupon rate
- Rate **zeroCoupon** (Time, int, bool extrapolate=false) const
zero-coupon rate

Dates

- virtual **Date** **todayDate** () const=0
today's date
- virtual **Date** **referenceDate** () const=0
the reference date, i.e., the date at which discount = 1
- virtual **DayCounter** **dayCounter** () const=0
the day counter used for date/time conversion
- virtual **Date** **maxDate** () const=0
the latest date for which the curve can return rates
- virtual Time **maxTime** () const
the latest time for which the curve can return rates

Protected Member Functions

Calculations

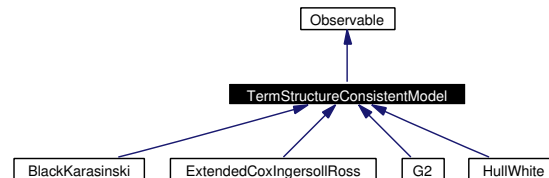
These methods must be implemented in derived classes to perform the actual discount and rate calculations

- virtual Rate **zeroYieldImpl** (Time, bool extrapolate=false) const=0
zero-yield calculation
- virtual DiscountFactor **discountImpl** (Time, bool extrapolate=false) const=0
discount calculation
- virtual Rate **forwardImpl** (Time, bool extrapolate=false) const=0
instantaneous forward-rate calculation
- virtual Rate **compoundForwardImpl** (Time, int, bool extrapolate=false) const=0
compound forward-rate calculation

9.393 TermStructureConsistentModel Class Reference

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

Inheritance diagram for TermStructureConsistentModel:



9.393.1 Detailed Description

Term-structure consistent model class.

This is a base class for models that can reprice exactly any discount bond.

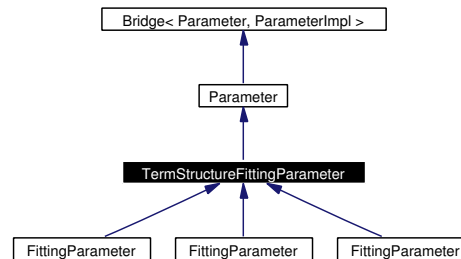
Public Member Functions

- `TermStructureConsistentModel` (const `RelinkableHandle< TermStructure >` &termStructure)
- `const RelinkableHandle< TermStructure > &termStructure () const`

9.394 TermStructureFittingParameter Class Reference

#include <ql/ShortRateModels/parameter.hpp>

Inheritance diagram for TermStructureFittingParameter:



9.394.1 Detailed Description

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

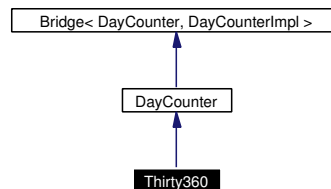
Public Member Functions

- `TermStructureFittingParameter` (const **Handle**< `Parameter::Impl` > &impl)
- `TermStructureFittingParameter` (const **RelinkableHandle**< `TermStructure` > &term)

9.395 Thirty360 Class Reference

```
#include <ql/DayCounters/thirty360.hpp>
```

Inheritance diagram for Thirty360:



9.395.1 Detailed Description

30/360 day count convention

The day count can be calculated according to US, European, or Italian conventions.

US (NASD) convention: if the starting date is the 31st of a month, it becomes equal to the 30th of the same month. If the ending date is the 31st of a month and the starting date is earlier than the 30th of a month, the ending date becomes equal to the 1st of the next month, otherwise the ending date becomes equal to the 30th of the same month.

European convention: starting dates or ending dates that occur on the 31st of a month become equal to the 30th of the same month.

Italian convention: starting dates or ending dates that occur on February and are greater than 27 become equal to 30 for computational sake.

Public Types

- enum **Convention** { **USA**, **European**, **Italian** }

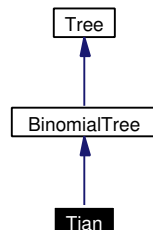
Public Member Functions

- **Thirty360** (Convention c=Thirty360::USA)

9.396 Tian Class Reference

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

Inheritance diagram for Tian:



9.396.1 Detailed Description

Tian tree: third moment matching, multiplicative approach

Public Member Functions

- **Tian** (const **Handle**< **DiffusionProcess** > &process, Time end, Size steps, double strike)
- double **underlying** (Size i, Size index) const
- double **probability** (Size, Size, Size) const

Protected Attributes

- double **up_**
- double **down_**
- double **pu_**
- double **pd_**

9.397 TimeBasket Class Reference

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

9.397.1 Detailed Description

Distribution over a number of dates.

Map interface

- typedef super::iterator **iterator**
- typedef super::const_iterator **const_iterator**
- typedef super::reverse_iterator **reverse_iterator**
- typedef super::const_reverse_iterator **const_reverse_iterator**
- bool **hasDate** (const **Date** &) const

membership

Public Member Functions

- **TimeBasket** (const std::vector< **Date** > &dates, const std::vector< double > &values)

Algebra

- **TimeBasket** & **operator+=** (const **TimeBasket** &other)
- **TimeBasket** & **operator-=** (const **TimeBasket** &other)

Other methods

- **TimeBasket** **rebin** (const std::vector< **Date** > &buckets) const
- redistribute the entries over the given dates*

9.398 TimeGrid Class Reference

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

9.398.1 Detailed Description

time grid class

Todo

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

Public Member Functions

- **TimeGrid** (Time end, Size steps)
Regularly spaced time-grid.
- `template<class Iterator> TimeGrid (Iterator begin, Iterator end)`
Time grid with mandatory time points.
- `template<class Iterator> TimeGrid (Iterator begin, Iterator end, Size steps)`
Time grid with mandatory time points.
- Size **findIndex** (Time t) const
- `const std::vector< Time > & mandatoryTimes () const`
- Time **dt** (Size i) const

9.398.2 Constructor & Destructor Documentation

9.398.2.1 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.

9.398.2.2 TimeGrid (Iterator *begin*, Iterator *end*, Size *steps*)

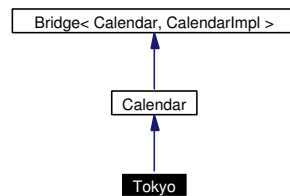
Time grid with mandatory time points.

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.

9.399 Tokyo Class Reference

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

Inheritance diagram for Tokyo:



9.399.1 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

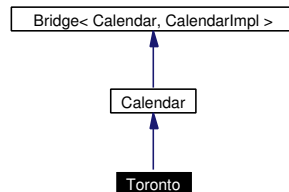
- a few one-shot holidays

Holidays falling on a Sunday are observed on the Monday following except for the Bank Holidays associated with the New Year

9.400 Toronto Class Reference

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

Inheritance diagram for Toronto:



9.400.1 Detailed Description

Toronto calendar

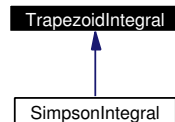
Holidays:

- 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)

9.401 TrapezoidIntegral Class Reference

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

Inheritance diagram for TrapezoidIntegral:



9.401.1 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.

Public Types

- enum **Method** { **Default**, **MidPoint** }

Public Member Functions

- **TrapezoidIntegral** (double accuracy, **Method** method=Default, Size maxIterations=**Null**<int >())
- template<class F> double **operator()** (const F &f, double a, double b) const
- double **accuracy** () const
- double & **accuracy** ()
- **Method** **method** () const
- **Method** & **method** ()
- Size **maxIterations** () const
- Size & **maxIterations** ()

Protected Member Functions

- template<class F> double **defaultIteration** (const F &f, double a, double b, double I, Size N) const
- template<class F> double **midPointIteration** (const F &f, double a, double b, double I, Size N) const

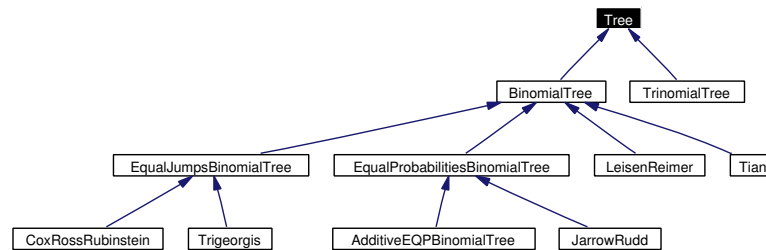
Protected Attributes

- double `accuracy_`
- Method `method_`
- Size `maxIterations_`

9.402 Tree Class Reference

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

Inheritance diagram for Tree:



9.402.1 Detailed Description

Tree approximating a single-factor diffusion

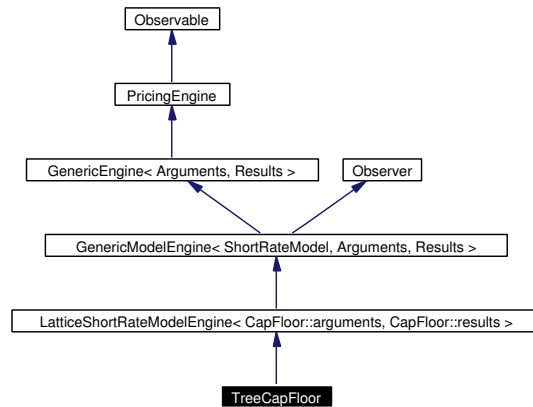
Public Member Functions

- **Tree** (Size nColumns)
- virtual double **underlying** (Size i, Size index) const=0
- virtual Size **size** (Size i) const=0
- virtual Size **descendant** (Size i, Size index, Size branch) const=0
- virtual double **probability** (Size i, Size index, Size branch) const=0
- Size **nColumns** () const

9.403 TreeCapFloor Class Reference

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

Inheritance diagram for TreeCapFloor:



9.403.1 Detailed Description

Cap/floor priced on a lattice.

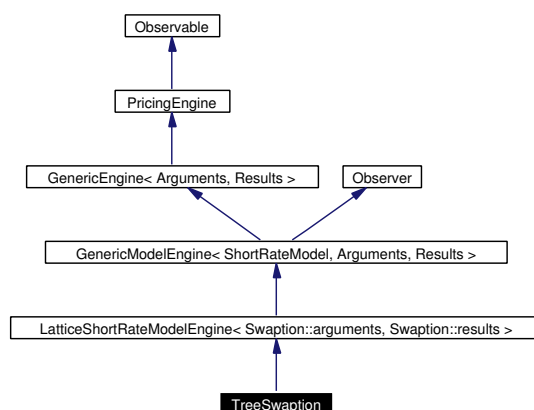
Public Member Functions

- **TreeCapFloor** (const **Handle**< **ShortRateModel** > &model, Size timeSteps)
- **TreeCapFloor** (const **Handle**< **ShortRateModel** > &model, const **TimeGrid** &timeGrid)
- void **calculate** () const

9.404 TreeSwaption Class Reference

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

Inheritance diagram for TreeSwaption:



9.404.1 Detailed Description

Swaption priced on a lattice

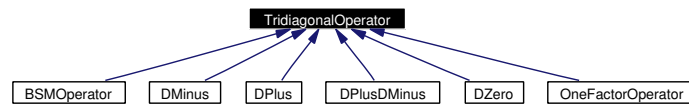
Public Member Functions

- **TreeSwaption** (const **Handle**< **ShortRateModel** > &model, Size timeSteps)
- **TreeSwaption** (const **Handle**< **ShortRateModel** > &model, const **TimeGrid** &timeGrid)
- void **calculate** () const

9.405 TridiagonalOperator Class Reference

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

Inheritance diagram for TridiagonalOperator:



9.405.1 Detailed Description

Base implementation for tridiagonal operator.

Warning:

to use real time-dependant algebra, you must overload the corresponding operators in the inheriting time-dependent class

Operator interface

- **Disposable< Array > applyTo** (const Array &v) const
apply operator to a given array
- **Disposable< Array > solveFor** (const Array &rhs) const
solve linear system for a given right-hand side
- **Disposable< Array > SOR** (const Array &rhs, double tol) const
solve linear system with SOR approach
- **Disposable< TridiagonalOperator > identity** (Size size)
identity instance

Public Types

- **typedef Array arrayType**

Public Member Functions

- **TridiagonalOperator** (Size size=0)
- **TridiagonalOperator** (const Array &low, const Array &mid, const Array &high)
- **TridiagonalOperator** (const Disposable< TridiagonalOperator > &)
- **TridiagonalOperator & operator=** (const Disposable< TridiagonalOperator > &)

Inspectors

- **Size size** () const

- **bool isTimeDependent ()**
- **const Array & lowerDiagonal () const**
- **const Array & diagonal () const**
- **const Array & upperDiagonal () const**

Modifiers

- **void setFirstRow (double, double)**
- **void setMidRow (Size, double, double, double)**
- **void setMidRows (double, double, double)**
- **void setLastRow (double, double)**
- **void setTime (Time t)**

Utilities

- **void swap (TridiagonalOperator &)**

Protected Attributes

- **Array diagonal_**
- **Array lowerDiagonal_**
- **Array upperDiagonal_**
- **Handle< TimeSetter > timeSetter_**

Friends

- **Disposable< TridiagonalOperator > operator+ (const TridiagonalOperator &)**
- **Disposable< TridiagonalOperator > operator- (const TridiagonalOperator &)**
- **Disposable< TridiagonalOperator > operator+ (const TridiagonalOperator &, const TridiagonalOperator &)**
- **Disposable< TridiagonalOperator > operator- (const TridiagonalOperator &, const TridiagonalOperator &)**
- **Disposable< TridiagonalOperator > operator * (double, const TridiagonalOperator &)**
- **Disposable< TridiagonalOperator > operator * (const TridiagonalOperator &, double)**
- **Disposable< TridiagonalOperator > operator/ (const TridiagonalOperator &, double)**

9.406 TridiagonalOperator::TimeSetter Class Reference

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

9.406.1 Detailed Description

encapsulation of time-setting logic

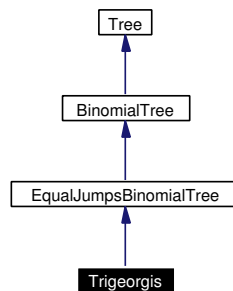
Public Member Functions

- virtual void **setTime** (Time t, **TridiagonalOperator** &L) const=0

9.407 Trigeorgis Class Reference

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

Inheritance diagram for Trigeorgis:



9.407.1 Detailed Description

Trigeorgis (additive equal jumps) binomial tree

Public Member Functions

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

9.408 TrinomialBranching Class Reference

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

9.408.1 Detailed Description

Branching scheme for a trinomial node.

Each node has three descendants, with the middle branch linked to the node which is closest to the expectation of the variable.

Public Member Functions

- Size **descendant** (Size index, Size branch) const
- double **probability** (Size index, Size branch) const
- int **jMin** () const

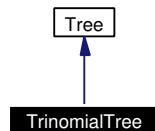
Friends

- class **TrinomialTree**

9.409 TrinomialTree Class Reference

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

Inheritance diagram for TrinomialTree:



9.409.1 Detailed Description

Recombining trinomial tree class.

This class defines a recombining trinomial tree approximating a diffusion.

Warning:

The diffusion term of the SDE must be independent of the underlying process.

Public Member Functions

- **TrinomialTree** (const **Handle**< **DiffusionProcess** > &process, const **TimeGrid** &timeGrid, bool isPositive=false)
- double **dx** (Size i) const
- Size **size** (Size i) const
- double **underlying** (Size i, Size index) const
- const **TimeGrid** & **timeGrid** () const
- Size **descendant** (Size i, Size index, Size branch) const
- double **probability** (Size i, Size index, Size branch) const

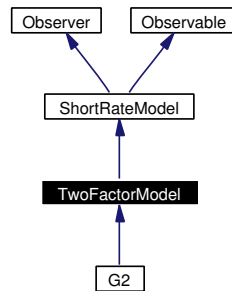
Protected Attributes

- std::vector< **Handle**< **TrinomialBranching** > > **branchings_**
- double **x0_**
- std::vector< double > **dx_**
- **TimeGrid** **timeGrid_**

9.410 TwoFactorModel Class Reference

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

Inheritance diagram for TwoFactorModel:



9.410.1 Detailed Description

Abstract base-class for two-factor models.

Public Member Functions

- **TwoFactorModel** (Size nParams)
- virtual **Handle**< **ShortRateDynamics** > **dynamics** () const=0
Returns the short-rate dynamics.
- virtual **Handle**< **Lattice** > **tree** (const **TimeGrid** &grid) const
Returns a two-dimensional trinomial tree.

9.411 TwoFactorModel::ShortRateDynamics Class Reference

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

9.411.1 Detailed Description

Class describing the dynamics of the two state variables.

We assume here that the short-rate is a function of two state variables x and y .

$$r_t = f(t, x_t, y_t)$$

of two state variables x_t and y_t . These stochastic processes satisfy

$$x_t = \mu_x(t, x_t)dt + \sigma_x(t, x_t)dW_t^x$$

and

$$y_t = \mu_y(t, y_t)dt + \sigma_y(t, y_t)dW_t^y$$

where W^x and W^y are two brownian motions satisfying

$$dW_t^x dW_t^y = \rho dt$$

.

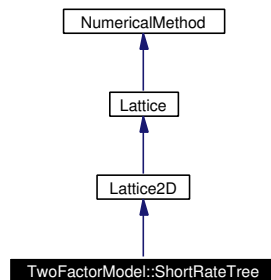
Public Member Functions

- **ShortRateDynamics** (const **Handle**< **DiffusionProcess** > &xProcess, const **Handle**< **DiffusionProcess** > &yProcess, double correlation)
- virtual Rate **shortRate** (Time t, double x, double y) const=0
- const **Handle**< **DiffusionProcess** > & **xProcess** () const
Risk-neutral dynamics of the first state variable x.
- const **Handle**< **DiffusionProcess** > & **yProcess** () const
Risk-neutral dynamics of the second state variable y.
- double **correlation** () const
Correlation ρ between the two brownian motions.

9.412 TwoFactorModel::ShortRateTree Class Reference

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

Inheritance diagram for TwoFactorModel::ShortRateTree:



9.412.1 Detailed Description

Recombining two-dimensional tree discretizing the state variable.

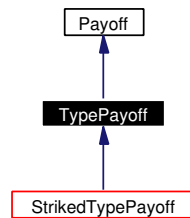
Public Member Functions

- **ShortRateTree** (const **Handle**< **TrinomialTree** > &tree1, const **Handle**< **TrinomialTree** > &tree2, const **Handle**< **ShortRateDynamics** > &dynamics)
Plain tree build-up from short-rate dynamics.
- DiscountFactor **discount** (Size i, Size index) const
Discount factor at time t_i and node indexed by index.

9.413 TypePayoff Class Reference

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

Inheritance diagram for TypePayoff:



9.413.1 Detailed Description

Intermediate class for call/put/straddle payoffs.

Public Member Functions

- **TypePayoff** (Option::Type type)
- Option::Type **optionType** () const

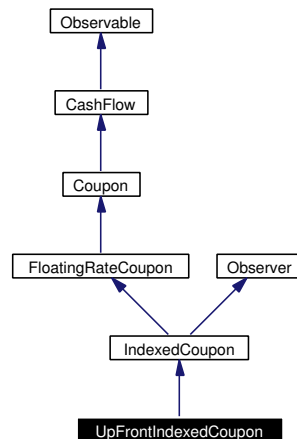
Protected Attributes

- Option::Type **type_**

9.414 UpFrontIndexedCoupon Class Reference

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

Inheritance diagram for UpFrontIndexedCoupon:



9.414.1 Detailed Description

up front indexed coupon class

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **UpFrontIndexedCoupon** (double nominal, const **Date** &paymentDate, const **Handle**<**Xibor** > &index, const **Date** &startDate, const **Date** &endDate, int fixingDays, Spread spread=0.0, const **Date** &refPeriodStart=**Date**(), const **Date** &refPeriodEnd=**Date**(), const **DayCounter** &dayCounter=**DayCounter**())

FloatingRateCoupon interface

- **Date** fixingDate () const

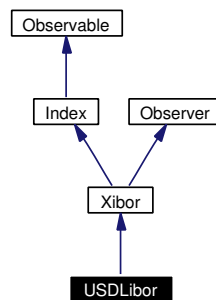
Visitability

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

9.415 USDLibor Class Reference

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

Inheritance diagram for USDLibor:



9.415.1 Detailed Description

USD Libor index

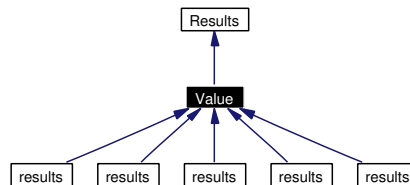
Public Member Functions

- **USDLibor** (int n, TimeUnit units, const **RelinkableHandle**< **TermStructure** > &h, const **DayCounter** &dc=**Actual360**())

9.416 Value Class Reference

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

Inheritance diagram for Value:



9.416.1 Detailed Description

pricing results

Public Member Functions

- void **reset** ()

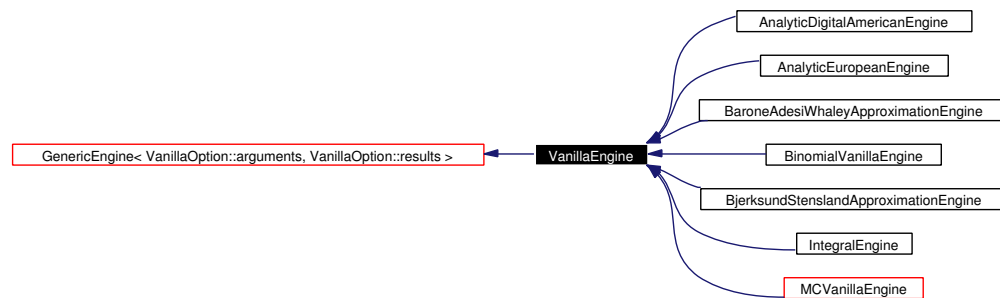
Public Attributes

- double **value**
- double **errorEstimate**

9.417 VanillaEngine Class Reference

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

Inheritance diagram for VanillaEngine:



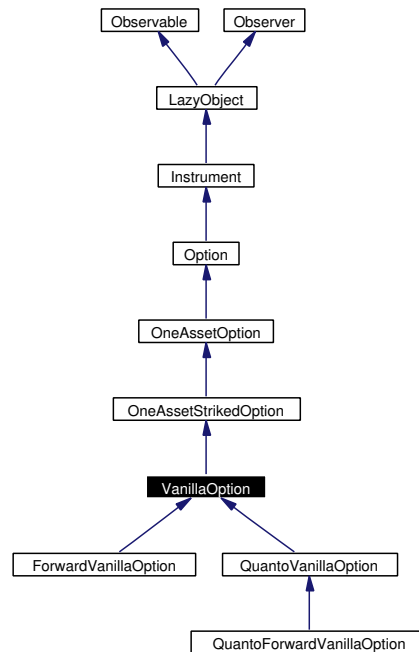
9.417.1 Detailed Description

Vanilla option engine base class.

9.418 VanillaOption Class Reference

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

Inheritance diagram for VanillaOption:



9.418.1 Detailed Description

Vanilla option (no discrete dividends, no barriers) on a single asset.

Public Member Functions

- **VanillaOption** (const **Handle**< BlackScholesStochasticProcess > &stochProc, const **Handle**< StrikedTypePayoff > &payoff, const **Handle**< Exercise > &exercise, const **Handle**< PricingEngine > &engine=**Handle**< PricingEngine >())

Protected Member Functions

- void **performCalculations** () const

9.418.2 Member Function Documentation

9.418.2.1 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.

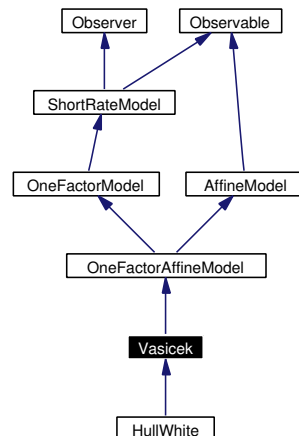
Reimplemented from **OneAssetStrikedOption** (p. [482](#)).

Reimplemented in **ForwardVanillaOption** (p. [316](#)), and **QuantoVanillaOption** (p. [531](#)).

9.419 Vasicek Class Reference

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

Inheritance diagram for Vasicek:



9.419.1 Detailed Description

Vasicek model class

This class implements the **Vasicek**(p. 626) model defined by

$$dr_t = a(b - r_t)dt + \sigma dW_t,$$

where a , b and σ are constants.

Public Member Functions

- **Vasicek** (Rate $r_0=0.05$, double $a=0.1$, double $b=0.05$, double $\sigma=0.01$)
- virtual double **discountBondOption** (Option::Type type, double strike, Time maturity, Time bondMaturity) const
- virtual **Handle**< ShortRateDynamics > **dynamics** () const
returns the short-rate dynamics

Protected Member Functions

- virtual double **A** (Time t , Time T) const
- virtual double **B** (Time t , Time T) const
- double **a** () const
- double **b** () const
- double **sigma** () const

9.420 Vasicek::Dynamics Class Reference

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

9.420.1 Detailed Description

Short-rate dynamics in the Vasicek model.

The short-rate follows an Ornstein-Uhlenbeck process with mean b .

Public Member Functions

- **Dynamics** (double a , double b , double σ , double r_0)
- virtual double **variable** (Time t , Rate r) const
- virtual double **shortRate** (Time t , double x) const

9.421 Visitor Class Template Reference

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

9.421.1 Detailed Description

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

Visitor for a specific class

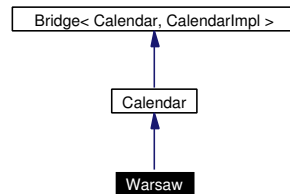
Public Member Functions

- virtual void **visit** (T &)=0

9.422 Warsaw Class Reference

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

Inheritance diagram for Warsaw:



9.422.1 Detailed Description

Warsaw calendar

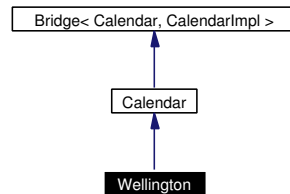
Holidays:

- 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

9.423 Wellington Class Reference

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

Inheritance diagram for Wellington:



9.423.1 Detailed Description

Wellington calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday or Tuesday)
- Day after New Year's Day, January 2st (possibly moved to Monday or Tuesday)
- Anniversary Day, Monday nearest January 22nd
- Waitangi Day. February 6th
- Good Friday
- Easter Monday
- ANZAC Day. April 25th
- Queen's Birthday, first Monday in June
- Labour Day, fourth Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

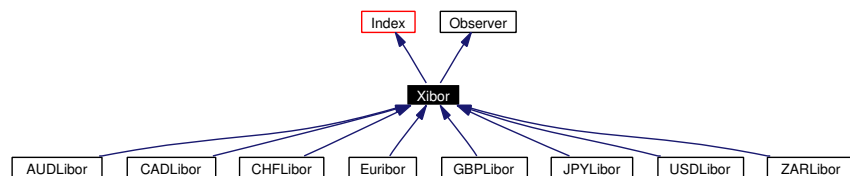
Note:

The holiday rules for **Wellington**(p. 630) were documented by David Gilbert for IDB (<http://www.jrefinery.com/ibd/>)

9.424 Xibor Class Reference

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

Inheritance diagram for Xibor:



9.424.1 Detailed Description

base class for libor indexes

Public Member Functions

- **Xibor** (const std::string &familyName, int n, TimeUnit units, int settlementDays, Currency currency, const **Calendar** &calendar, bool isAdjusted, RollingConvention rollingConvention, const **DayCounter** &dayCounter, const **RelinkableHandle**< **TermStructure** > &h)

Index interface

- Rate **fixing** (const **Date** &fixingDate) const
returns the fixing at the given date

Observer interface

- void **update** ()

Inspectors

- std::string **name** () const
Returns the name of the index.
- **Period tenor** () const
- int **frequency** () const
- int **settlementDays** () const
- Currency **currency** () const
- **Calendar calendar** () const
- bool **isAdjusted** () const
- RollingConvention **rollingConvention** () const
- **DayCounter dayCounter** () const
- **Handle**< **TermStructure** > **termStructure** () const

9.424.2 Member Function Documentation

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

returns the fixing at the given date

Note:

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

Implements **Index** (p. [363](#)).

9.424.2.2 `void update ()` [virtual]

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

Implements **Observer** (p. [475](#)).

9.424.2.3 `std::string name () const` [virtual]

Returns the name of the index.

Warning:

This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implements **Index** (p. [363](#)).

9.425 XiborManager Class Reference

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

9.425.1 Detailed Description

global repository for libor histories

Public Types

- typedef std::map< std::string, **History** > **HistoryMap**

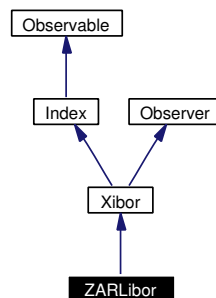
Static Public Member Functions

- void **setHistory** (const std::string &name, const **History** &)
- const **History** & **getHistory** (const std::string &name)
- bool **hasHistory** (const std::string &name)
- std::vector< std::string > **histories** ()

9.426 ZARLibor Class Reference

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

Inheritance diagram for ZARLibor:



9.426.1 Detailed Description

ZAR Libor index, also known as JIBAR

Todo

check settlement days

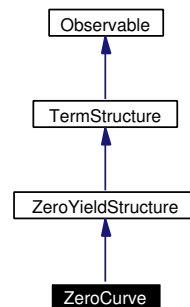
Public Member Functions

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

9.427 ZeroCurve Class Reference

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

Inheritance diagram for ZeroCurve:



9.427.1 Detailed Description

Term structure based on linear interpolation of zero yields.

Public Member Functions

- **ZeroCurve** (const **Date** &today'sDate, const std::vector< **Date** > &dates, const std::vector< Rate > &yields, const **DayCounter** &dayCounter=**Actual365**())
- **DayCounter** dayCounter () const
the day counter used for date/time conversion
- **Date** today'sDate () const
today's date
- **Date** referenceDate () const
the reference date, i.e., the date at which discount = 1
- const std::vector< **Date** > & **dates** () const
- **Date** maxDate () const
the latest date for which the curve can return rates
- const std::vector< Time > & **times** () const
- Time maxTime () const
the latest time for which the curve can return rates

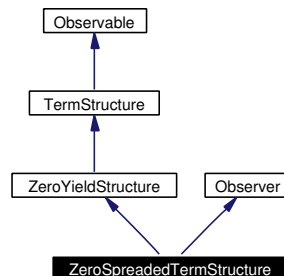
Protected Member Functions

- Rate **zeroYieldImpl** (Time t, bool extrapolate=false) const
zero-yield calculation

9.428 ZeroSpreadedTermStructure Class Reference

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

Inheritance diagram for ZeroSpreadedTermStructure:



9.428.1 Detailed Description

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

Note:

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

Public Member Functions

- **ZeroSpreadedTermStructure** (const **RelinkableHandle**< **TermStructure** > &, const **RelinkableHandle**< **Quote** > &spread)

TermStructure interface

- **DayCounter** **dayCounter** () const
the day counter used for date/time conversion
- **Date** **referenceDate** () const
the reference date, i.e., the date at which discount = 1
- **Date** **todayDate** () const
today's date
- **Date** **maxDate** () const
the latest date for which the curve can return rates
- **Time** **maxTime** () const
the latest time for which the curve can return rates

Observer interface

- void **update** ()

Protected Member Functions

- Rate **zeroYieldImpl** (Time, bool extrapolate=false) const
returns the spreaded zero yield rate
- Rate **forwardImpl** (Time, bool extrapolate=false) const
returns the spreaded forward rate

9.428.2 Member Function Documentation

9.428.2.1 void update () [virtual]

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

Implements **Observer** (p. [475](#)).

9.428.2.2 Rate forwardImpl (Time, bool *extrapolate* = false) const [protected, virtual]

returns the spreaded forward rate

Warning:

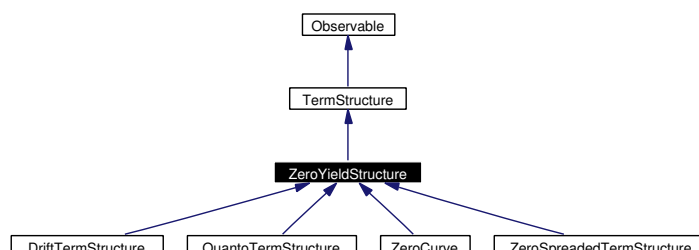
This method must disappear should the spread become a curve

Reimplemented from **ZeroYieldStructure** (p. [638](#)).

9.429 ZeroYieldStructure Class Reference

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

Inheritance diagram for ZeroYieldStructure:



9.429.1 Detailed Description

Zero yield term structure.

This abstract class acts as an adapter to **TermStructure**(p. 594) allowing the programmer to implement only the `zeroYieldImpl(Time, bool)` method in derived classes.

Rates are assumed to be annual continuous compounding.

Protected Member Functions

- DiscountFactor **discountImpl** (Time, bool extrapolate=false) const
- Rate **forwardImpl** (Time, bool extrapolate=false) const
- Rate **compoundForwardImpl** (Time, int, bool extrapolate=false) const

9.429.2 Member Function Documentation

9.429.2.1 DiscountFactor **discountImpl** (Time, bool *extrapolate* = false) const [protected, virtual]

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

Implements **TermStructure** (p. 595).

9.429.2.2 Rate **forwardImpl** (Time, bool *extrapolate* = false) const [protected, virtual]

Returns the instantaneous forward rate for the given date calculating it from the zero yield.

Implements **TermStructure** (p. 595).

Reimplemented in **ZeroSpreadTermStructure** (p. 637).

9.429.2.3 Rate compoundForwardImpl (Time, int, bool *extrapolate* = false) const [protected, virtual]

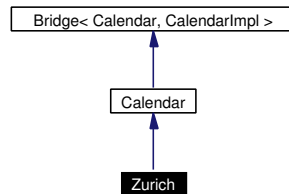
Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Implements **TermStructure** (p. [595](#)).

9.430 Zurich Class Reference

```
#include <ql/Calendars/zurich.hpp>
```

Inheritance diagram for Zurich:



9.430.1 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 10

QuantLib File Documentation

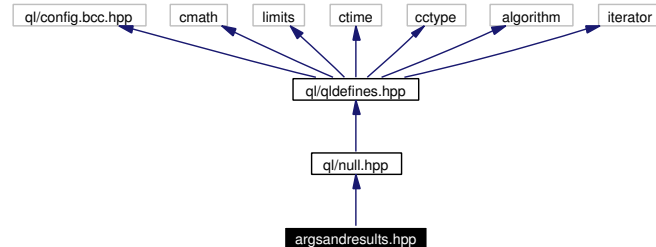
10.1 ql/argsandresults.hpp File Reference

10.1.1 Detailed Description

Base classes for generic arguments and results.

```
#include <ql/null.hpp>
```

Include dependency graph for argsandresults.hpp:



Namespaces

- namespace **QuantLib**

Defines

- `#define QL_MIN_VOLATILITY 0.0`
- `#define QL_MIN_DIVYIELD 1.0e-7`
- `#define QL_MAX_VOLATILITY 4.0`
- `#define QL_MAX_DIVYIELD 4.0`

10.2 ql/calendar.hpp File Reference

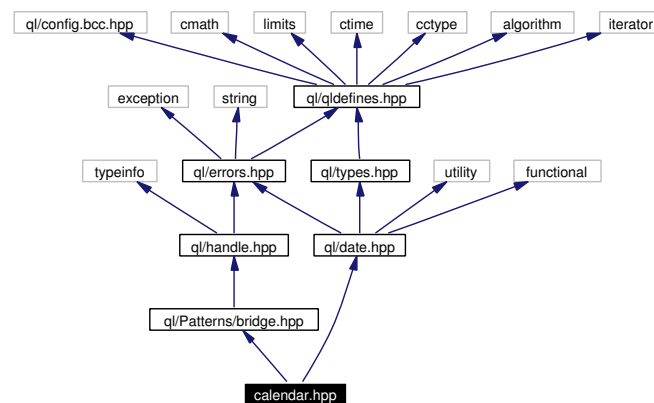
10.2.1 Detailed Description

calendar class

```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/bridge.hpp>
```

Include dependency graph for calendar.hpp:



Namespaces

- namespace **QuantLib**

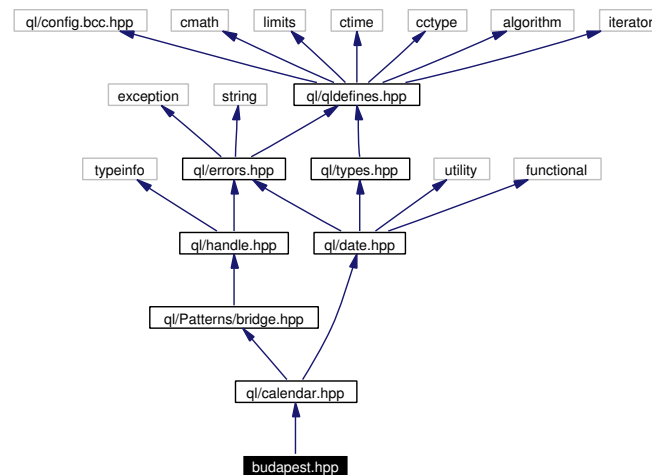
10.3 ql/Calendars/budapest.hpp File Reference

10.3.1 Detailed Description

Budapest calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for budapest.hpp:



Namespaces

- namespace **QuantLib**

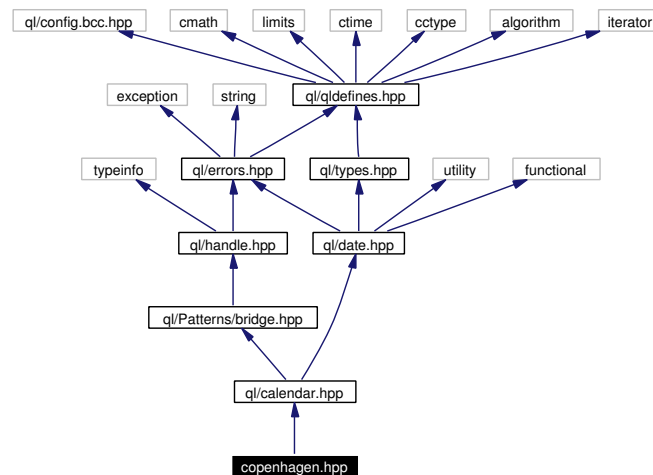
10.4 ql/Calendars/copenhagen.hpp File Reference

10.4.1 Detailed Description

Copenhagen calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for copenhagen.hpp:



Namespaces

- namespace **QuantLib**

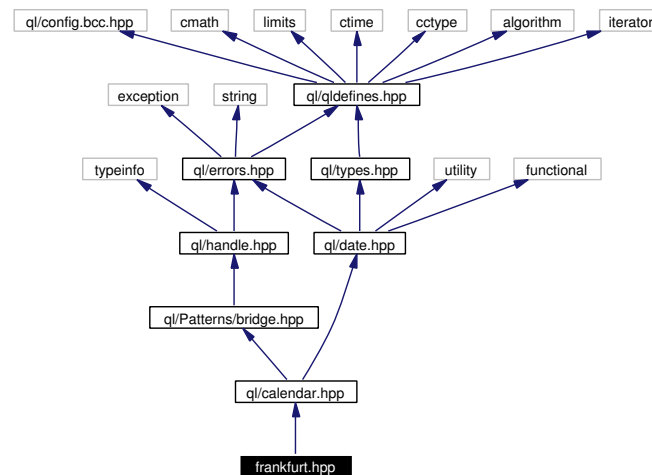
10.5 ql/Calendars/frankfurt.hpp File Reference

10.5.1 Detailed Description

Frankfurt calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for frankfurt.hpp:



Namespaces

- namespace **QuantLib**

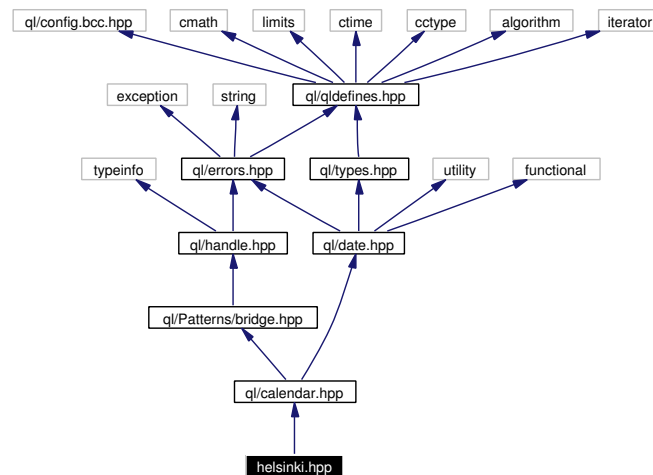
10.6 ql/Calendars/helsinki.hpp File Reference

10.6.1 Detailed Description

Helsinki calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for helsinki.hpp:



Namespaces

- namespace **QuantLib**

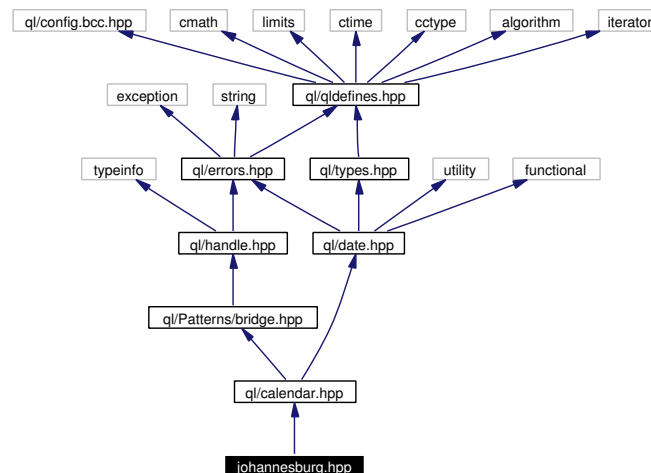
10.7 ql/Calendars/johannesburg.hpp File Reference

10.7.1 Detailed Description

Johannesburg calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for johannesburg.hpp:



Namespaces

- namespace **QuantLib**

10.8 ql/Calendars/jointcalendar.hpp File Reference

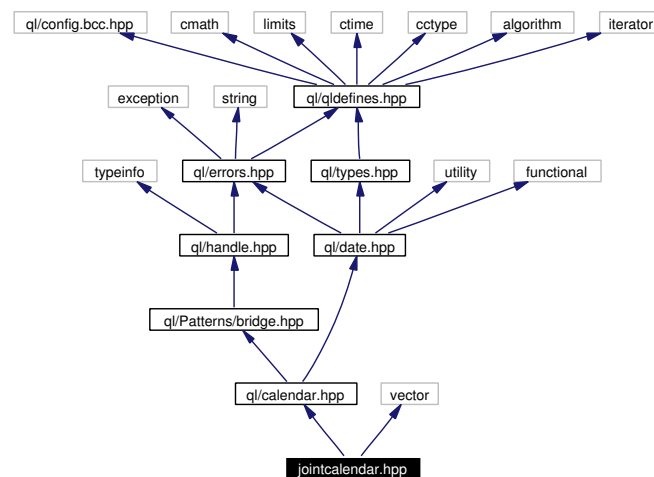
10.8.1 Detailed Description

Joint calendar.

```
#include <ql/calendar.hpp>
```

```
#include <vector>
```

Include dependency graph for jointcalendar.hpp:



Namespaces

- namespace **QuantLib**

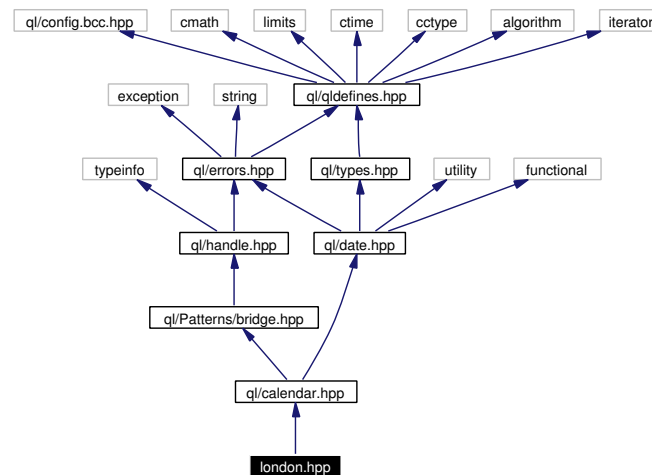
10.9 ql/Calendars/london.hpp File Reference

10.9.1 Detailed Description

London calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for london.hpp:



Namespaces

- namespace **QuantLib**

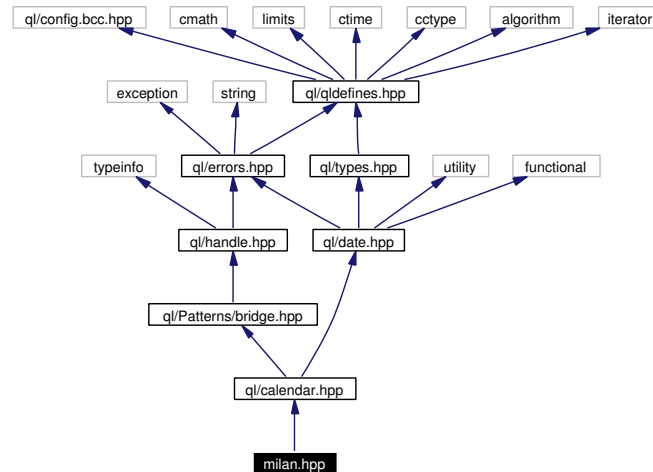
10.10 ql/Calendars/milan.hpp File Reference

10.10.1 Detailed Description

Milan calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for milan.hpp:



Namespaces

- namespace **QuantLib**

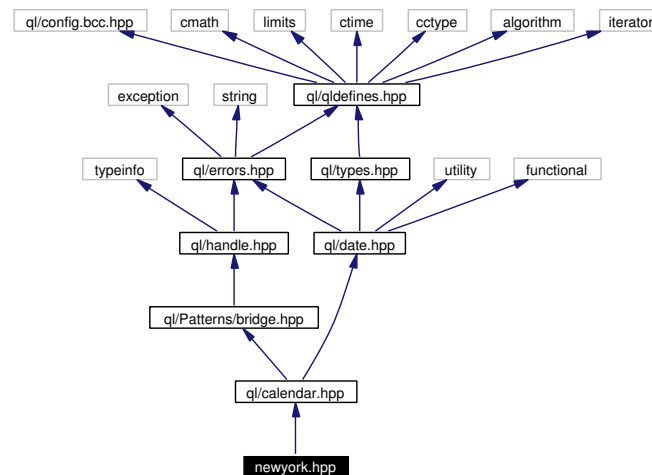
10.11 ql/Calendars/newyork.hpp File Reference

10.11.1 Detailed Description

New York calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for newyork.hpp:



Namespaces

- namespace **QuantLib**

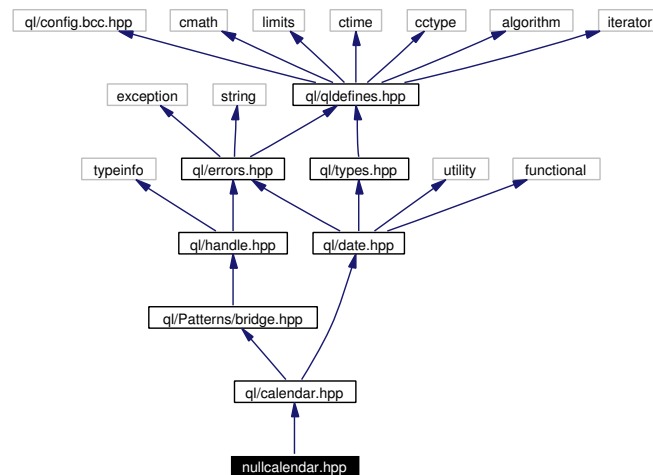
10.12 ql/Calendars/nullcalendar.hpp File Reference

10.12.1 Detailed Description

Calendar for reproducing theoretical calculations.

```
#include <ql/calendar.hpp>
```

Include dependency graph for nullcalendar.hpp:



Namespaces

- namespace **QuantLib**

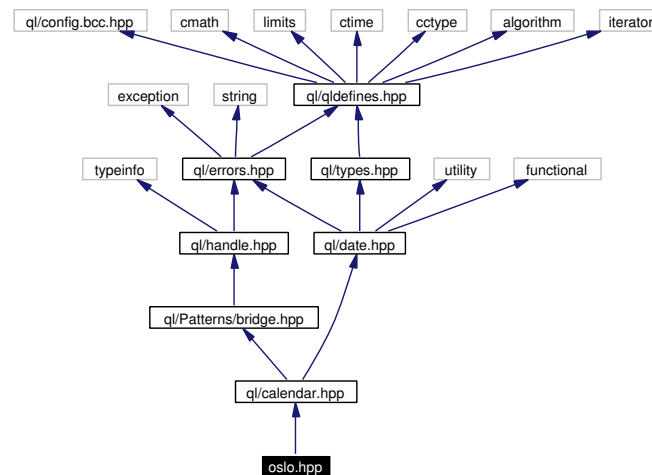
10.13 ql/Calendars/oslo.hpp File Reference

10.13.1 Detailed Description

Oslo calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for oslo.hpp:



Namespaces

- namespace **QuantLib**

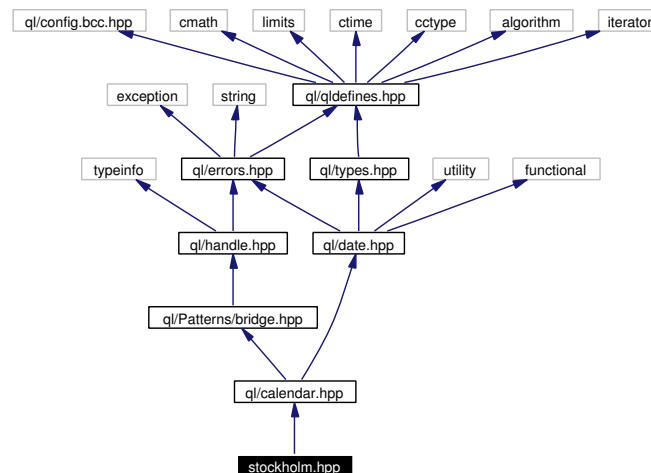
10.14 ql/Calendars/stockholm.hpp File Reference

10.14.1 Detailed Description

Stockholm calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for stockholm.hpp:



Namespaces

- namespace **QuantLib**

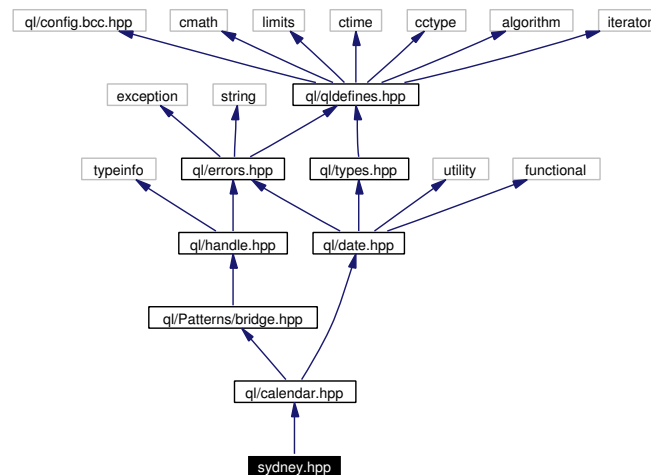
10.15 ql/Calendars/sydney.hpp File Reference

10.15.1 Detailed Description

Sydney calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for sydney.hpp:



Namespaces

- namespace **QuantLib**

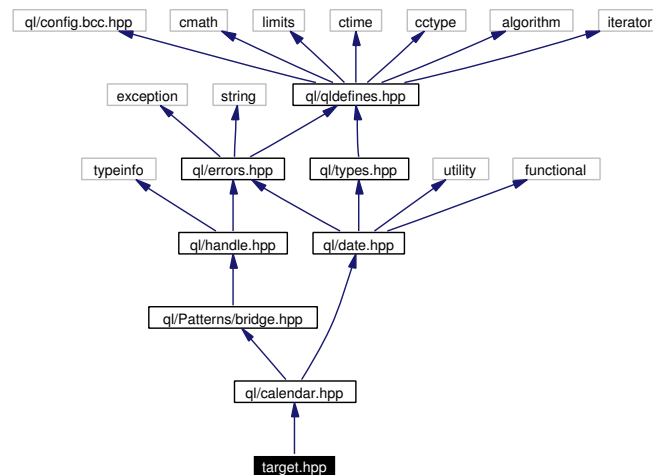
10.16 ql/Calendars/target.hpp File Reference

10.16.1 Detailed Description

TARGET calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for target.hpp:



Namespaces

- namespace **QuantLib**

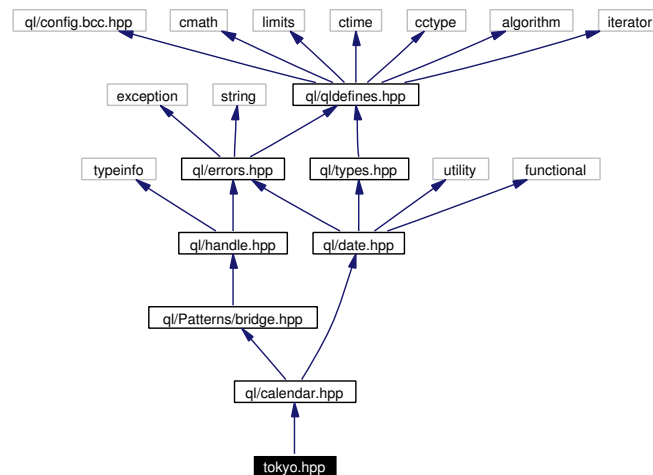
10.17 ql/Calendars/tokyo.hpp File Reference

10.17.1 Detailed Description

Tokyo calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for tokyo.hpp:



Namespaces

- namespace **QuantLib**

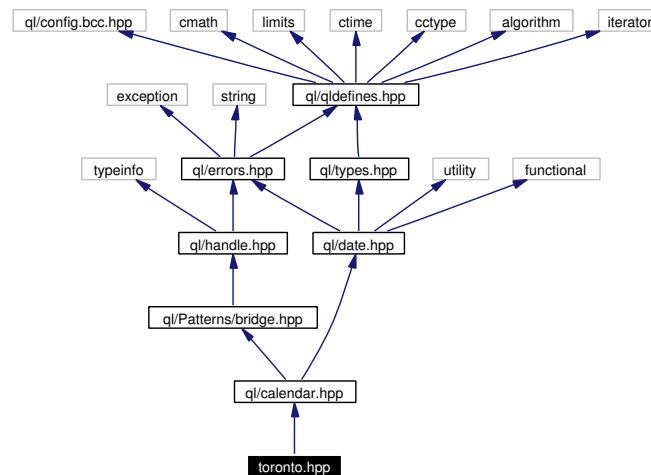
10.18 ql/Calendars/toronto.hpp File Reference

10.18.1 Detailed Description

Toronto calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for toronto.hpp:



Namespaces

- namespace **QuantLib**

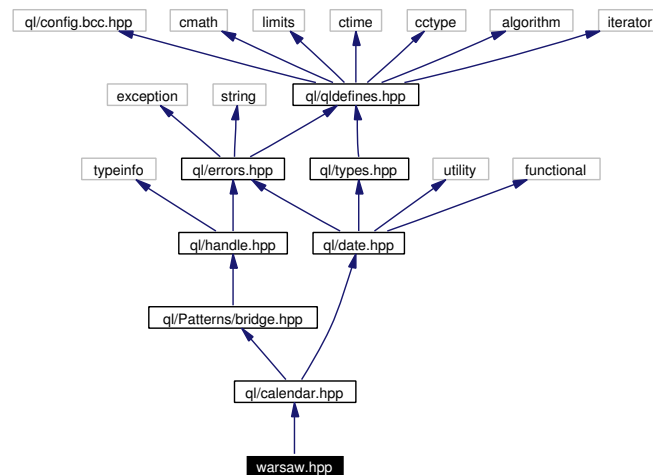
10.19 ql/Calendars/warsaw.hpp File Reference

10.19.1 Detailed Description

Warsaw calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for warsaw.hpp:



Namespaces

- namespace **QuantLib**

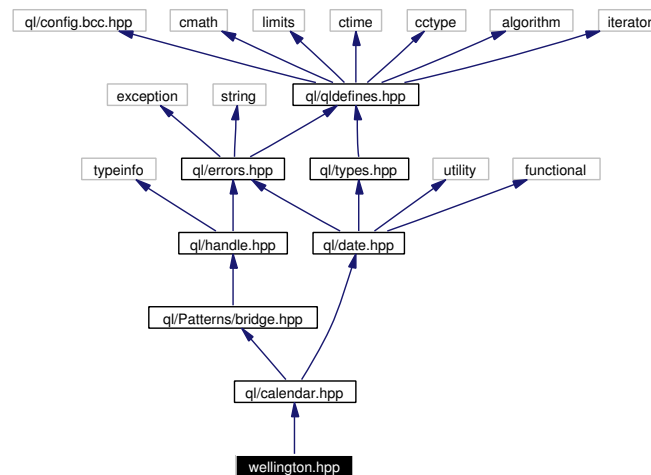
10.20 ql/Calendars/wellington.hpp File Reference

10.20.1 Detailed Description

Wellington calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for wellington.hpp:



Namespaces

- namespace **QuantLib**

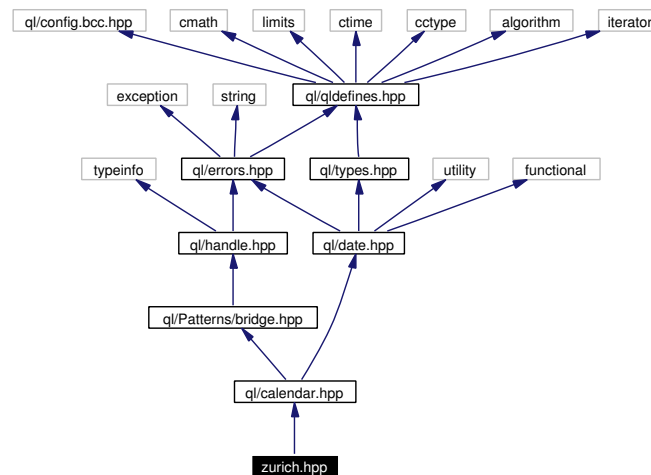
10.21 ql/Calendars/zurich.hpp File Reference

10.21.1 Detailed Description

Zurich calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for zurich.hpp:



Namespaces

- namespace **QuantLib**

10.22 ql/capvolstructures.hpp File Reference

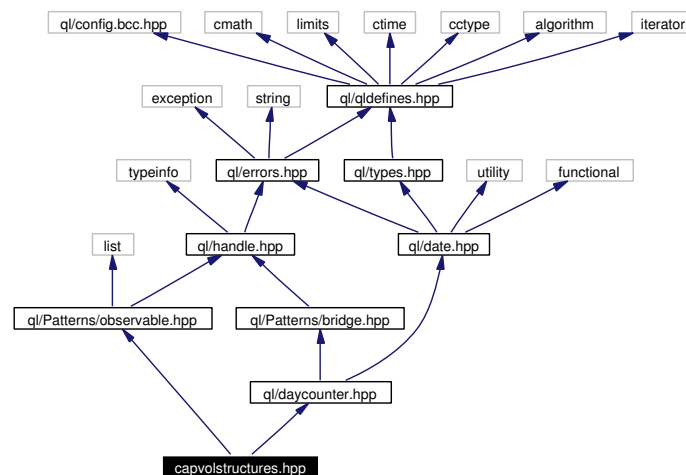
10.22.1 Detailed Description

Cap/Floor volatility structures.

```
#include <ql/daycounter.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for capvolstructures.hpp:



Namespaces

- namespace **QuantLib**

10.23 ql/cashflow.hpp File Reference

10.23.1 Detailed Description

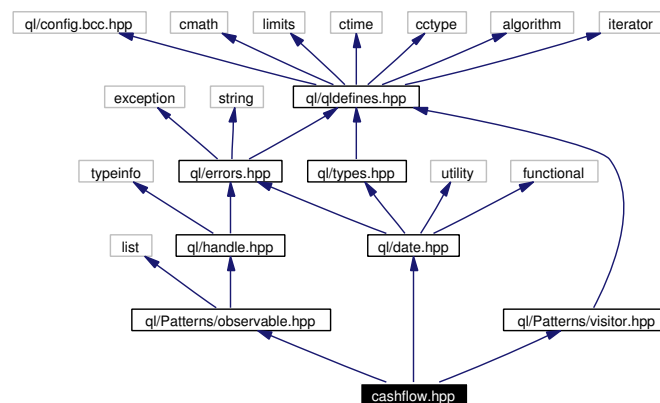
Base class for cash flows.

```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

```
#include <ql/Patterns/visitor.hpp>
```

Include dependency graph for cashflow.hpp:



Namespaces

- namespace **QuantLib**

10.24 ql/CashFlows/basispointsensitivity.hpp File Reference

10.24.1 Detailed Description

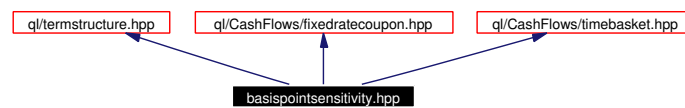
basis point sensitivity calculator

```
#include <ql/termstructure.hpp>
```

```
#include <ql/CashFlows/fixedratecoupon.hpp>
```

```
#include <ql/CashFlows/timebasket.hpp>
```

Include dependency graph for basispointsensitivity.hpp:



Namespaces

- namespace **QuantLib**

10.25 ql/CashFlows/cashflowvectors.hpp File Reference

10.25.1 Detailed Description

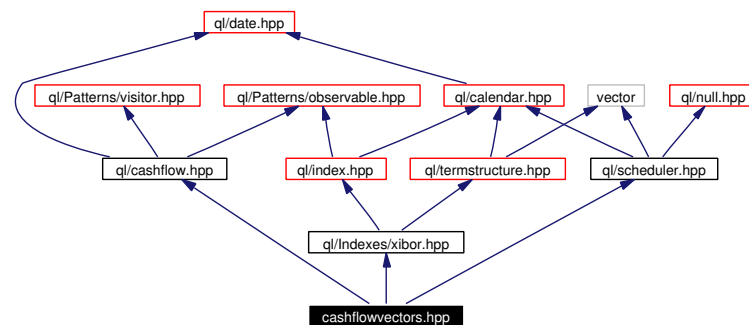
Cash flow vector builders.

```
#include <ql/cashflow.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/scheduler.hpp>
```

Include dependency graph for cashflowvectors.hpp:



Namespaces

- namespace **QuantLib**

10.26 ql/CashFlows/coupon.hpp File Reference

10.26.1 Detailed Description

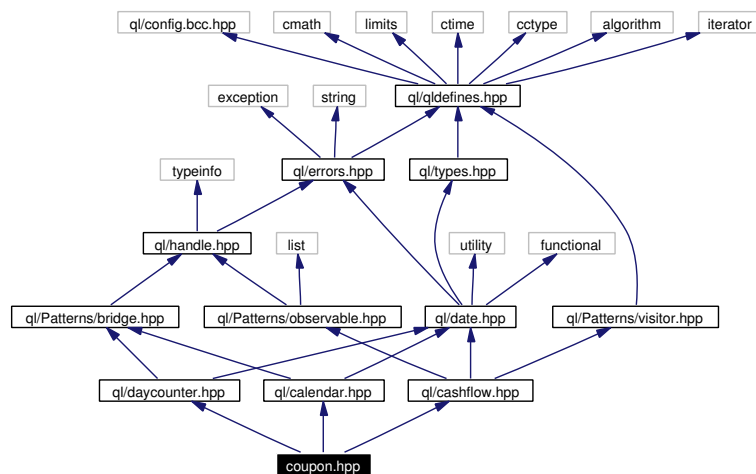
Coupon accruing over a fixed period.

```
#include <ql/cashflow.hpp>
```

```
#include <ql/calendar.hpp>
```

```
#include <ql/daycounter.hpp>
```

Include dependency graph for coupon.hpp:



Namespaces

- namespace **QuantLib**

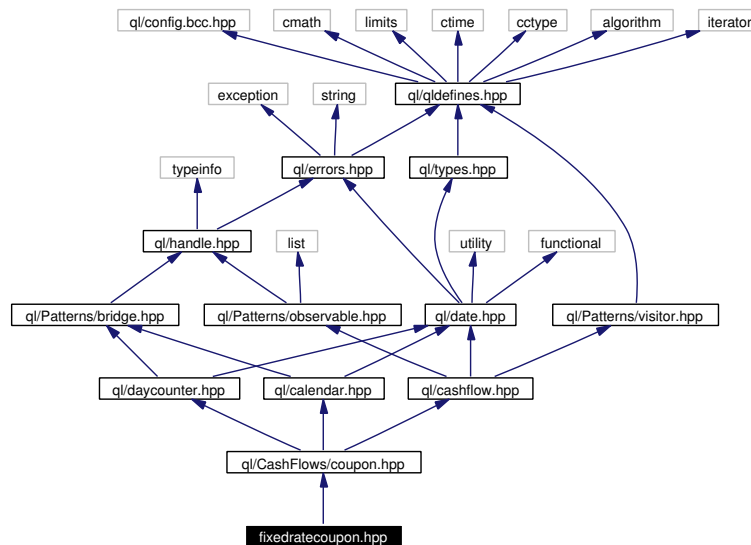
10.27 ql/CashFlows/fixedratecoupon.hpp File Reference

10.27.1 Detailed Description

Coupon paying a fixed annual rate.

```
#include <ql/CashFlows/coupon.hpp>
```

Include dependency graph for fixedratecoupon.hpp:



Namespaces

- namespace **QuantLib**

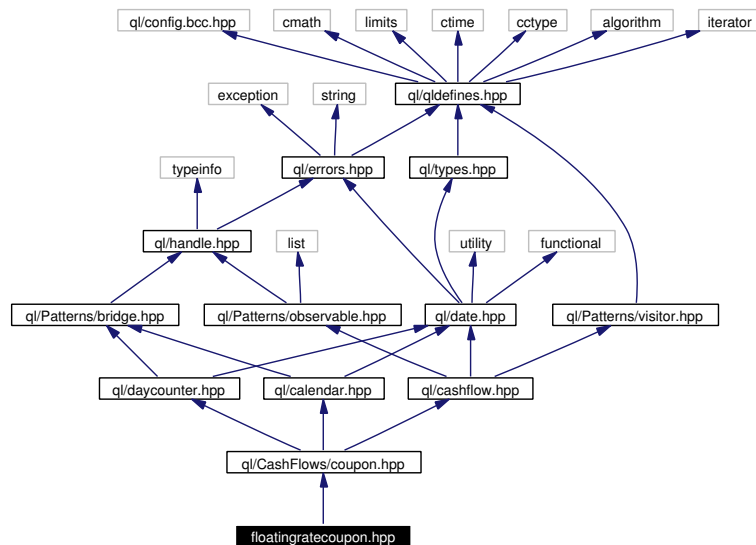
10.28 ql/CashFlows/floatingratecoupon.hpp File Reference

10.28.1 Detailed Description

Coupon at par on a term structure.

```
#include <ql/CashFlows/coupon.hpp>
```

Include dependency graph for floatingratecoupon.hpp:



Namespaces

- namespace **QuantLib**

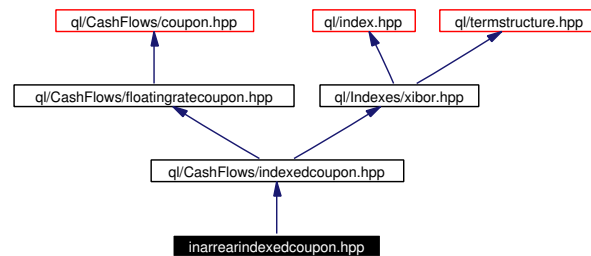
10.29 ql/CashFlows/inarrearindexedcoupon.hpp File Reference

10.29.1 Detailed Description

in arrear indexed coupon

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

Include dependency graph for inarrearindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**

10.30 ql/CashFlows/indexcashflowvectors.hpp File Reference

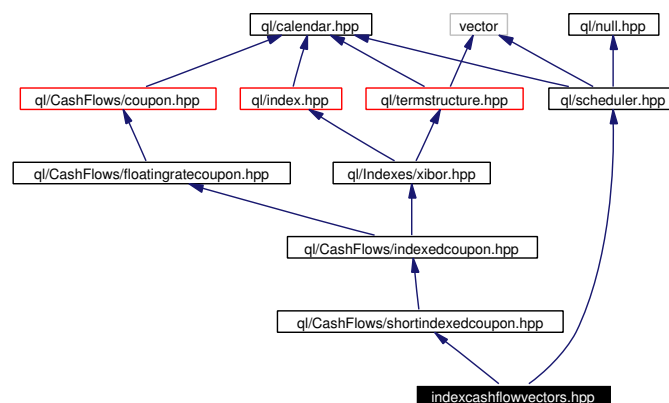
10.30.1 Detailed Description

Index Cash flow vector builders.

```
#include <ql/CashFlows/shortindexedcoupon.hpp>
```

```
#include <ql/scheduler.hpp>
```

Include dependency graph for indexcashflowvectors.hpp:



Namespaces

- namespace **QuantLib**

10.31 ql/CashFlows/indexedcoupon.hpp File Reference

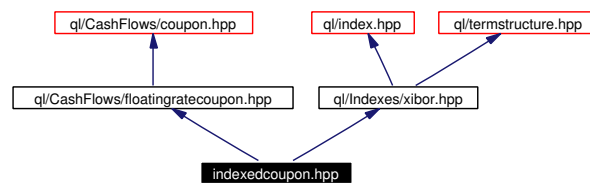
10.31.1 Detailed Description

indexed coupon

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for indexedcoupon.hpp:



Namespaces

- namespace **QuantLib**

10.32 ql/CashFlows/parcoupon.hpp File Reference

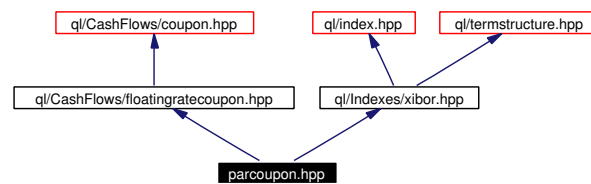
10.32.1 Detailed Description

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**

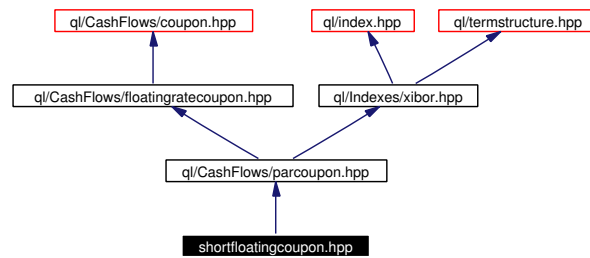
10.33 ql/CashFlows/shortfloatingcoupon.hpp File Reference

10.33.1 Detailed Description

Short (or long) coupon at par on a term structure.

```
#include <ql/CashFlows/parcoupon.hpp>
```

Include dependency graph for shortfloatingcoupon.hpp:



Namespaces

- namespace **QuantLib**

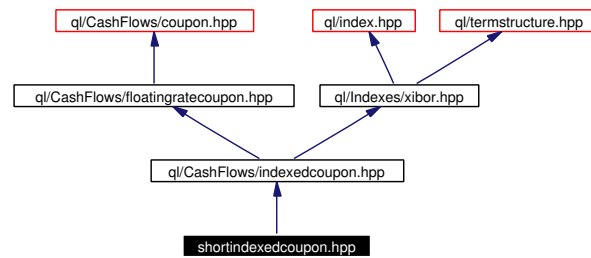
10.34 ql/CashFlows/shortindexedcoupon.hpp File Reference

10.34.1 Detailed Description

Short (or long) indexed coupon.

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

Include dependency graph for shortindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**

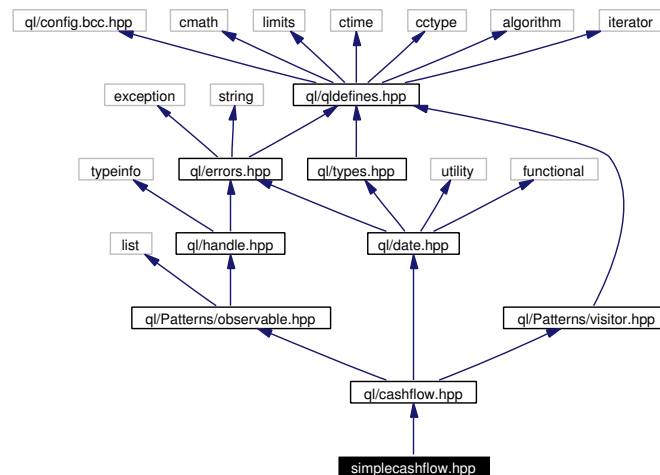
10.35 ql/CashFlows/simplecashflow.hpp File Reference

10.35.1 Detailed Description

Predetermined cash flow.

```
#include <ql/cashflow.hpp>
```

Include dependency graph for simplecashflow.hpp:



Namespaces

- namespace **QuantLib**

10.36 ql/CashFlows/timebasket.hpp File Reference

10.36.1 Detailed Description

Distribution over a number of dates

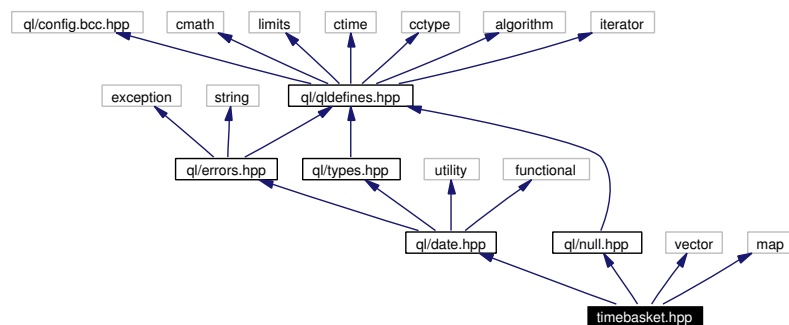
```
#include <ql/date.hpp>
```

```
#include <ql/null.hpp>
```

```
#include <vector>
```

```
#include <map>
```

Include dependency graph for timebasket.hpp:



Namespaces

- namespace **QuantLib**

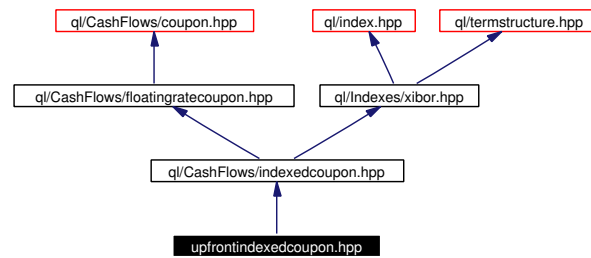
10.37 ql/CashFlows/upfrontindexedcoupon.hpp File Reference

10.37.1 Detailed Description

Up front indexed coupon.

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

Include dependency graph for upfrontindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**

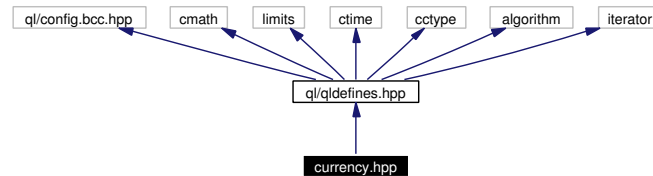
10.38 ql/currency.hpp File Reference

10.38.1 Detailed Description

Known currencies.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for currency.hpp:



Namespaces

- namespace **QuantLib**

10.39 ql/dataformatters.hpp File Reference

10.39.1 Detailed Description

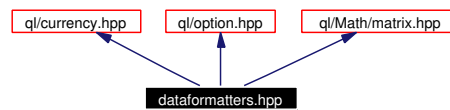
Classes used to format data for output.

```
#include <ql/currency.hpp>
```

```
#include <ql/option.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for dataformatters.hpp:



Namespaces

- namespace **QuantLib**

10.40 ql/dataparsers.hpp File Reference

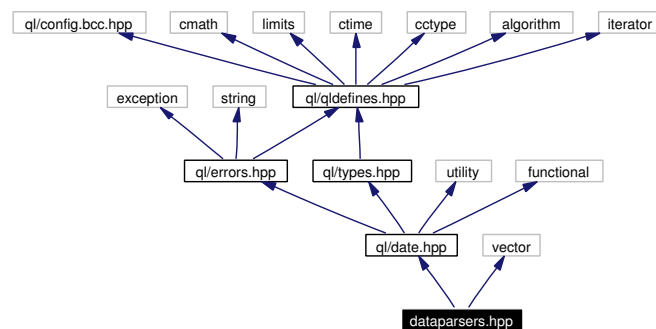
10.40.1 Detailed Description

Classes used to parse data for input.

```
#include <ql/date.hpp>
```

```
#include <vector>
```

Include dependency graph for dataparsers.hpp:



Namespaces

- namespace **QuantLib**

10.41 ql/date.hpp File Reference

10.41.1 Detailed Description

date- and time-related classes, typedefs and enumerations

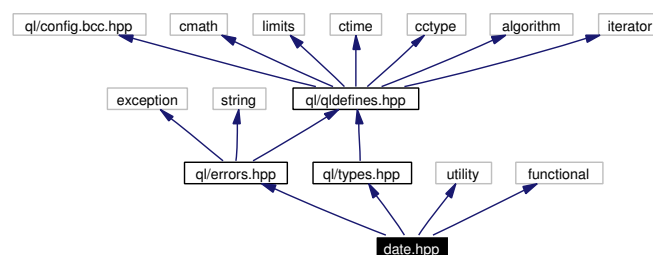
```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <utility>
```

```
#include <functional>
```

Include dependency graph for date.hpp:



Namespaces

- namespace **QuantLib**

10.42 ql/daycounter.hpp File Reference

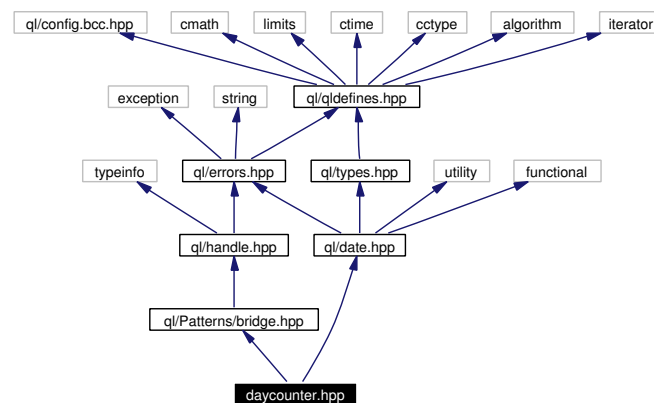
10.42.1 Detailed Description

day counter class

```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/bridge.hpp>
```

Include dependency graph for daycounter.hpp:



Namespaces

- namespace **QuantLib**

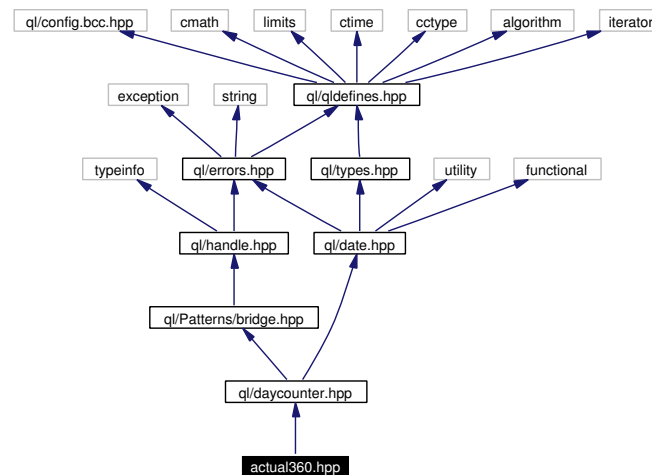
10.43 ql/DayCounters/actual360.hpp File Reference

10.43.1 Detailed Description

act/360 day counter

```
#include <ql/daycounter.hpp>
```

Include dependency graph for actual360.hpp:



Namespaces

- namespace **QuantLib**

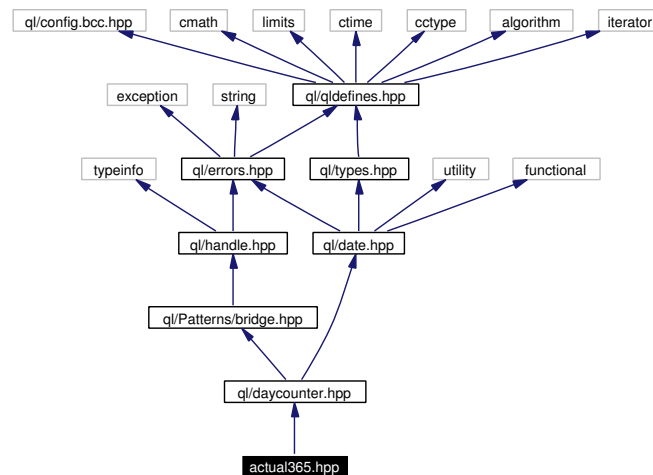
10.44 ql/DayCounters/actual365.hpp File Reference

10.44.1 Detailed Description

act/365 day counter

```
#include <ql/daycounter.hpp>
```

Include dependency graph for actual365.hpp:



Namespaces

- namespace **QuantLib**

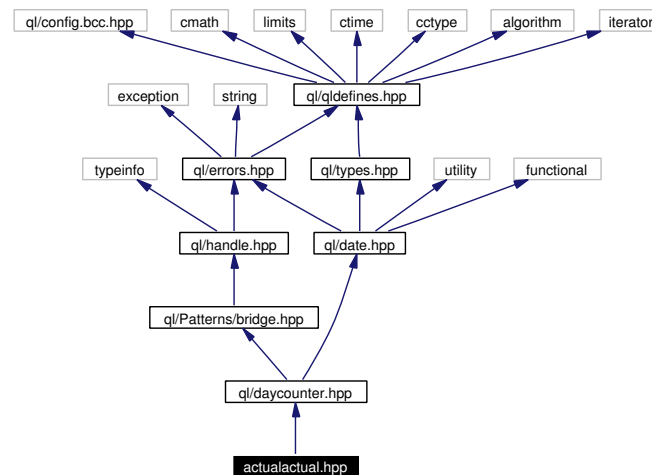
10.45 ql/DayCounters/actualactual.hpp File Reference

10.45.1 Detailed Description

act/act day counters

```
#include <ql/daycounter.hpp>
```

Include dependency graph for actualactual.hpp:



Namespaces

- namespace **QuantLib**

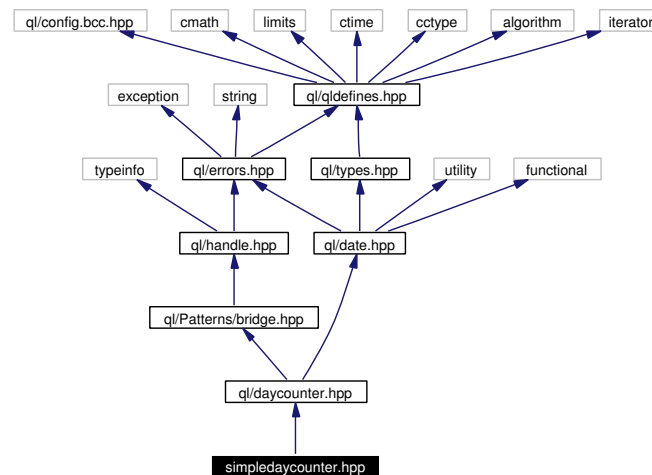
10.46 ql/DayCounters/simpliedaycounter.hpp File Reference

10.46.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

```
#include <ql/daycounter.hpp>
```

Include dependency graph for `simpliedaycounter.hpp`:



Namespaces

- namespace **QuantLib**

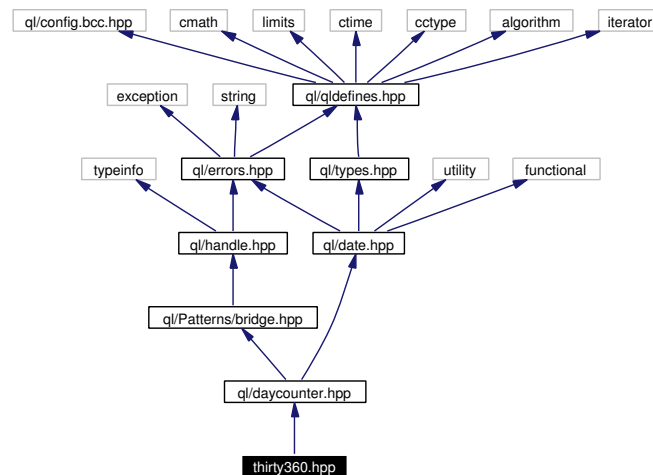
10.47 ql/DayCounters/thirty360.hpp File Reference

10.47.1 Detailed Description

30/360 day counters

```
#include <ql/daycounter.hpp>
```

Include dependency graph for thirty360.hpp:



Namespaces

- namespace **QuantLib**

10.48 ql/diffusionprocess.hpp File Reference

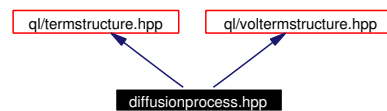
10.48.1 Detailed Description

Diffusion process.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for diffusionprocess.hpp:



Namespaces

- namespace **QuantLib**

10.49 ql/discretizedasset.hpp File Reference

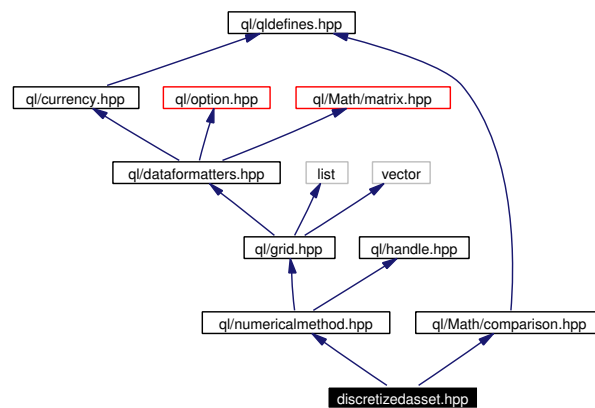
10.49.1 Detailed Description

Discretized asset classes.

```
#include <ql/numericalmethod.hpp>
```

```
#include <ql/Math/comparison.hpp>
```

Include dependency graph for discretizedasset.hpp:



Namespaces

- namespace **QuantLib**

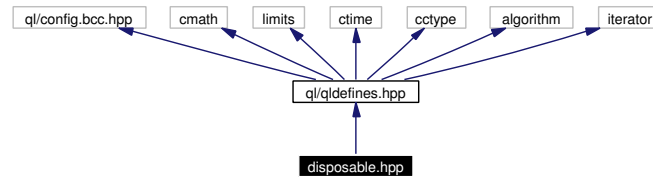
10.50 ql/disposable.hpp File Reference

10.50.1 Detailed Description

generic disposable object with move semantics

```
#include <ql/qldefines.hpp>
```

Include dependency graph for disposable.hpp:



Namespaces

- namespace **QuantLib**

10.51 ql/errors.hpp File Reference

10.51.1 Detailed Description

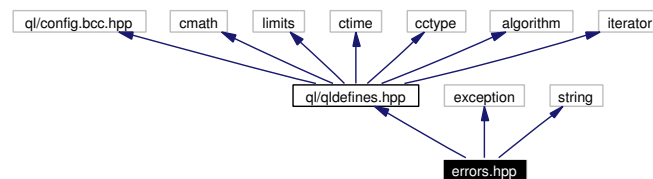
Classes and functions for error handling.

```
#include <ql/qldefines.hpp>
```

```
#include <exception>
```

```
#include <string>
```

Include dependency graph for errors.hpp:



Namespaces

- namespace **QuantLib**

Defines

- **#define QL_FAIL(message)**
throw an error (possibly with file and line information)
- **#define QL_ASSERT(condition, description)**
throw an error if the given condition is not verified
- **#define QL_REQUIRE(condition, description)**
throw an error if the given pre-condition is not verified
- **#define QL_ENSURE(condition, description)**
throw an error if the given post-condition is not verified

10.51.2 Define Documentation

10.51.2.1 #define QL_FAIL(message)

Value:

```
throw QuantLib::Error(\
    QuantLib::Error::where(__FILE__, __LINE__) + message)
```

throw an error (possibly with file and line information)

10.51.2.2 **#define QL_ASSERT(condition, description)**

Value:

```
if (!(condition)) \  
    throw QuantLib::AssertionFailedError(\  
        QuantLib::Error::where(__FILE__, __LINE__) + description); \  
else
```

throw an error if the given condition is not verified

10.51.2.3 **#define QL_REQUIRE(condition, description)**

Value:

```
if (!(condition)) \  
    throw QuantLib::PreconditionNotSatisfiedError(\  
        QuantLib::Error::where(__FILE__, __LINE__) + description); \  
else
```

throw an error if the given pre-condition is not verified

Examples:

DiscreteHedging.cpp, and **swapvaluation.cpp**.

10.51.2.4 **#define QL_ENSURE(condition, description)**

Value:

```
if (!(condition)) \  
    throw QuantLib::PostconditionNotSatisfiedError(\  
        QuantLib::Error::where(__FILE__, __LINE__) + description); \  
else
```

throw an error if the given post-condition is not verified

10.52 ql/exercise.hpp File Reference

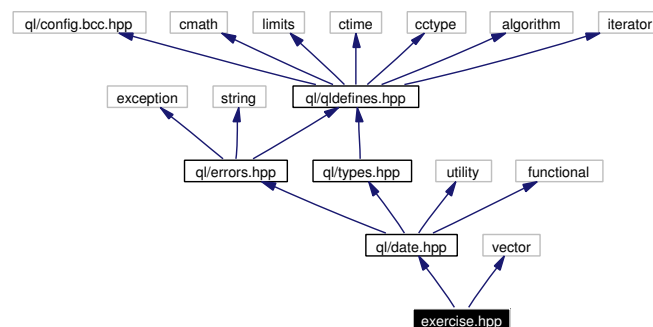
10.52.1 Detailed Description

Option exercise classes and payoff function.

```
#include <ql/date.hpp>
```

```
#include <vector>
```

Include dependency graph for exercise.hpp:



Namespaces

- namespace **QuantLib**

10.53 ql/FiniteDifferences/americancondition.hpp File Reference

10.53.1 Detailed Description

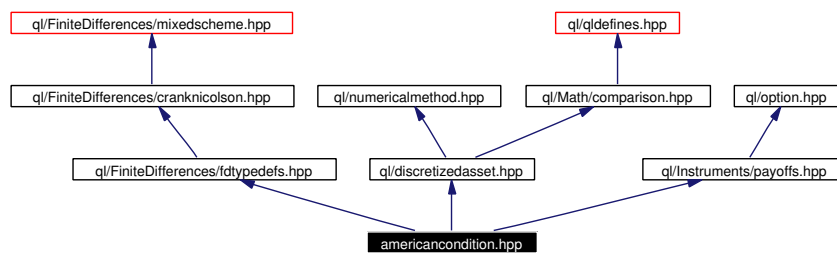
american option exercise condition

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for americancondition.hpp:



Namespaces

- namespace **QuantLib**

10.54 ql/FiniteDifferences/boundarycondition.hpp File Reference

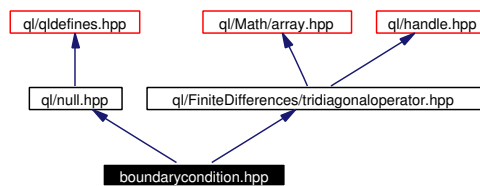
10.54.1 Detailed Description

boundary conditions for differential operators

```
#include <ql/null.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for boundarycondition.hpp:



Namespaces

- namespace **QuantLib**

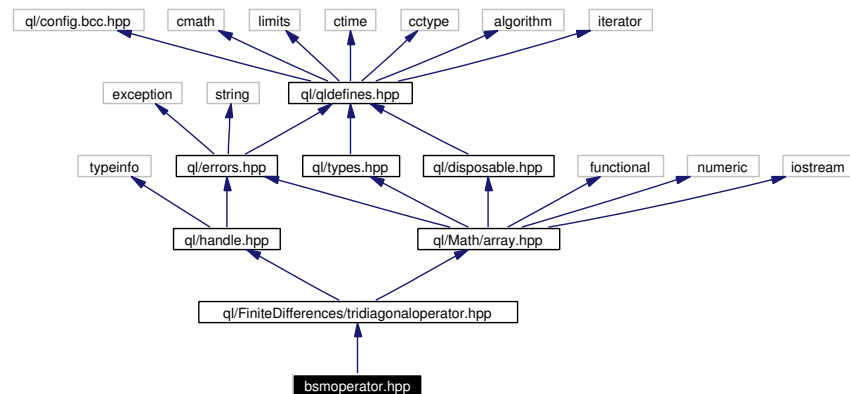
10.55 ql/FiniteDifferences/bsmoperator.hpp File Reference

10.55.1 Detailed Description

differential operator for Black-Scholes-Merton equation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for bsmoperator.hpp:



Namespaces

- namespace **QuantLib**

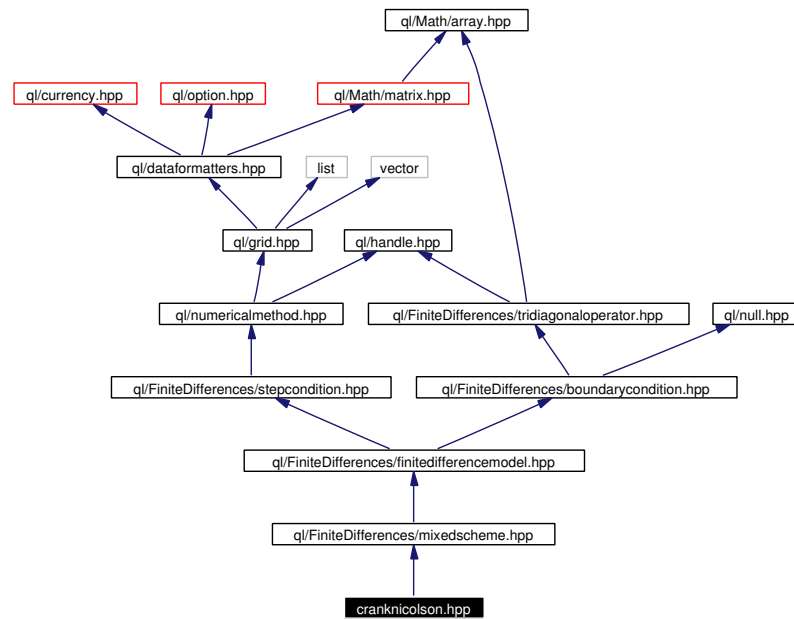
10.56 ql/FiniteDifferences/cranknicolson.hpp File Reference

10.56.1 Detailed Description

Crank-Nicolson scheme for finite difference methods.

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for cranknicolson.hpp:



Namespaces

- namespace **QuantLib**

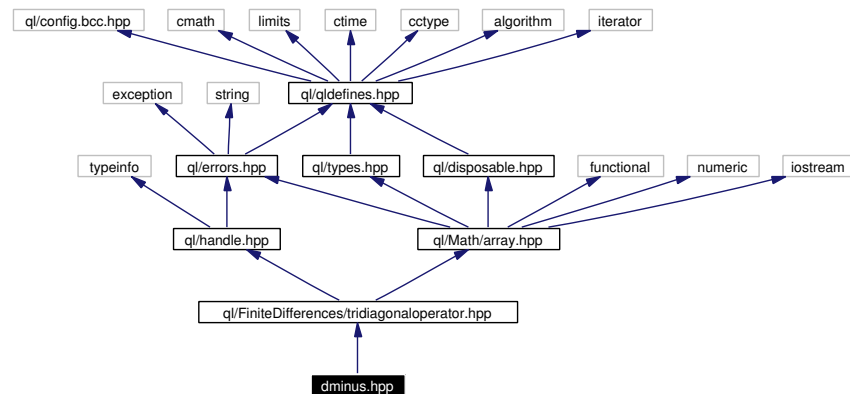
10.57 ql/FiniteDifferences/dminus.hpp File Reference

10.57.1 Detailed Description

*D*_- matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dminus.hpp:



Namespaces

- namespace **QuantLib**

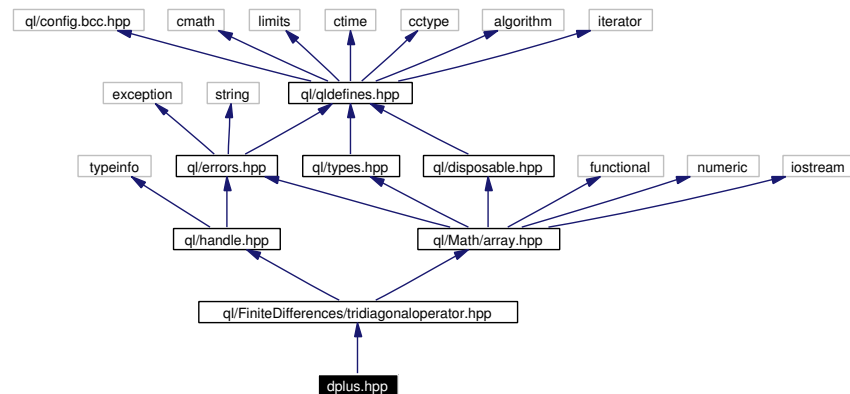
10.58 ql/FiniteDifferences/dplus.hpp File Reference

10.58.1 Detailed Description

D_+ matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dplus.hpp:



Namespaces

- namespace **QuantLib**

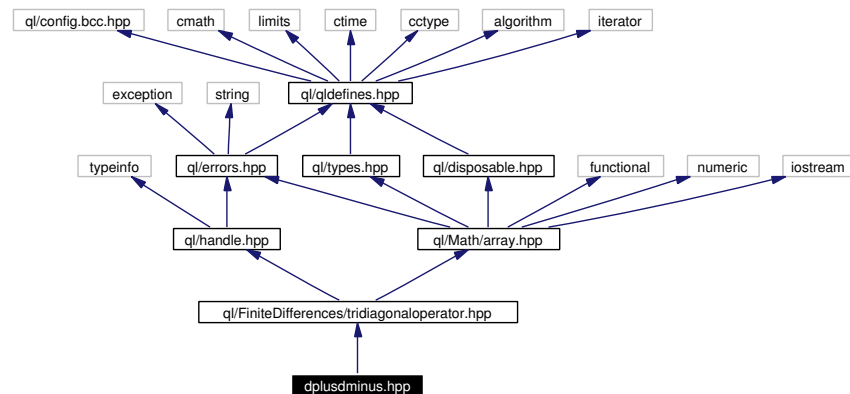
10.59 ql/FiniteDifferences/dplusdminus.hpp File Reference

10.59.1 Detailed Description

D_+D_- matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dplusdminus.hpp:



Namespaces

- namespace **QuantLib**

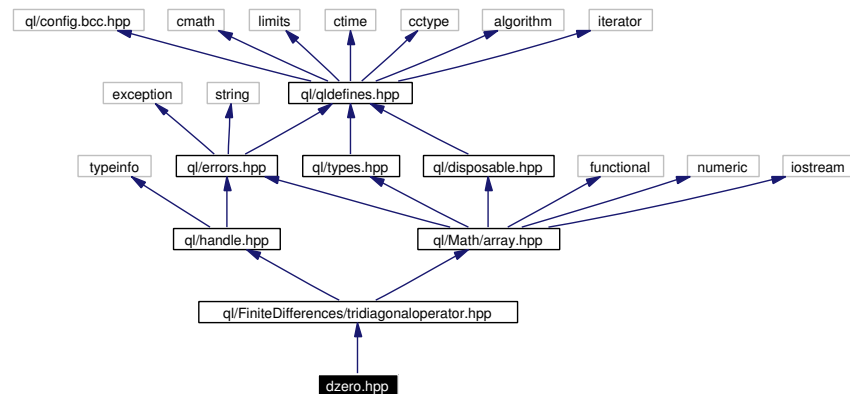
10.60 ql/FiniteDifferences/dzero.hpp File Reference

10.60.1 Detailed Description

D_0 matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dzero.hpp:



Namespaces

- namespace **QuantLib**

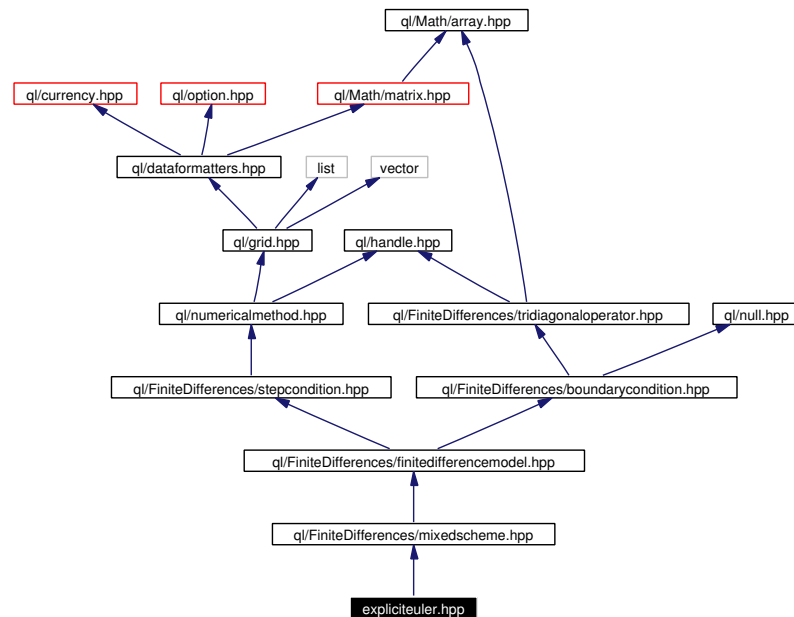
10.61 ql/FiniteDifferences/expliciteuler.hpp File Reference

10.61.1 Detailed Description

explicit Euler scheme for finite difference methods

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for expliciteuler.hpp:



Namespaces

- namespace **QuantLib**

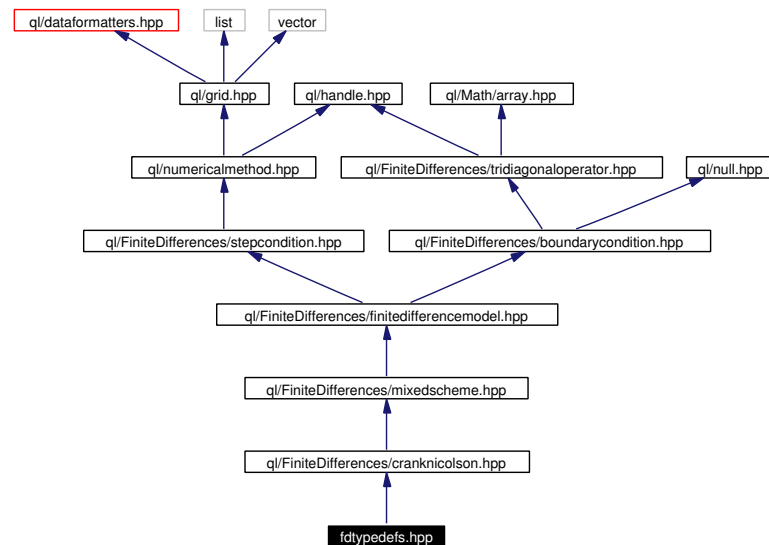
10.62 ql/FiniteDifferences/fdtypedefs.hpp File Reference

10.62.1 Detailed Description

default choices for template instantiations

```
#include <ql/FiniteDifferences/cranknicolson.hpp>
```

Include dependency graph for fdtypedefs.hpp:



Namespaces

- namespace **QuantLib**

10.63 ql/FiniteDifferences/finitedifferencemodel.hpp File Reference

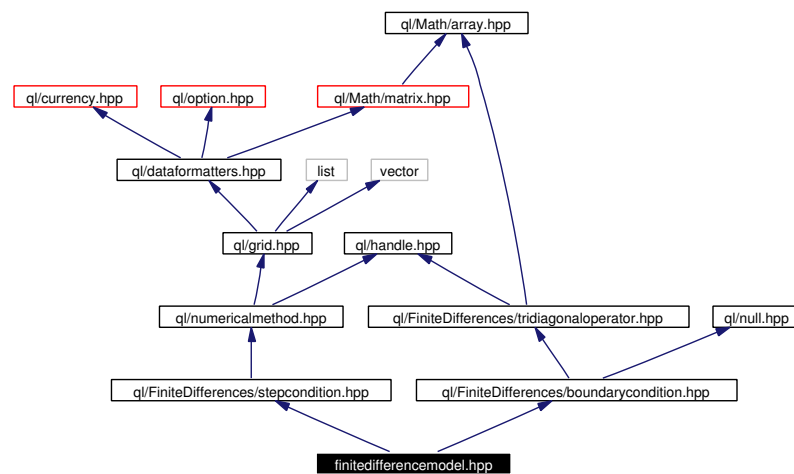
10.63.1 Detailed Description

generic finite difference model

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Include dependency graph for finitedifferencemodel.hpp:



Namespaces

- namespace **QuantLib**

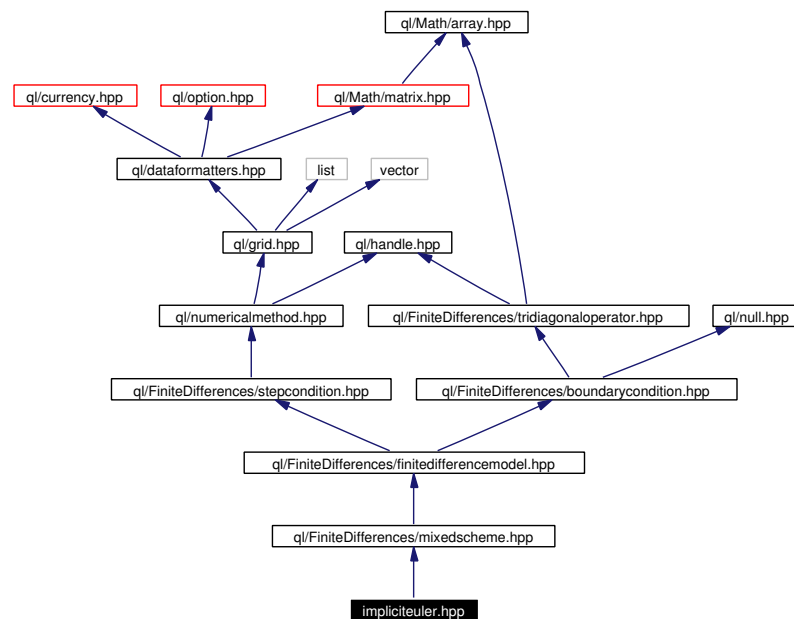
10.64 ql/FiniteDifferences/impliciteuler.hpp File Reference

10.64.1 Detailed Description

implicit Euler scheme for finite difference methods

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for impliciteuler.hpp:



Namespaces

- namespace **QuantLib**

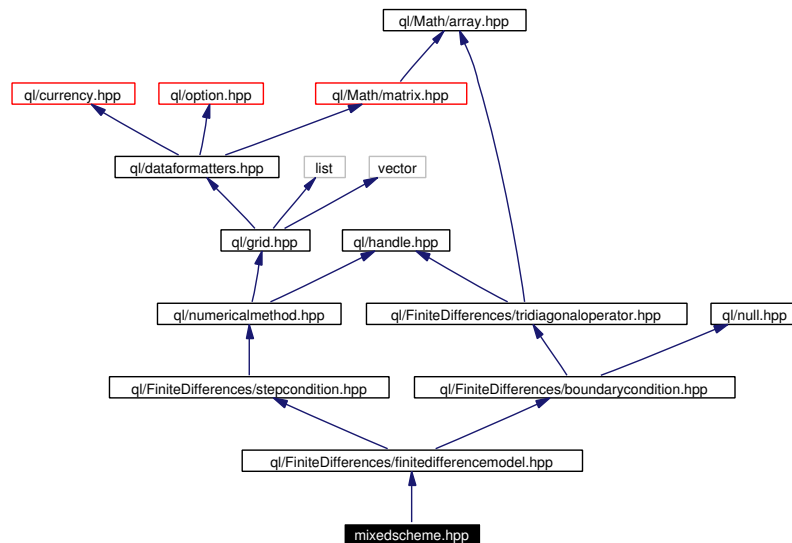
10.65 ql/FiniteDifferences/mixedscheme.hpp File Reference

10.65.1 Detailed Description

Mixed (explicit/implicit) scheme for finite difference methods.

```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
```

Include dependency graph for mixedscheme.hpp:



Namespaces

- namespace **QuantLib**

10.66 ql/FiniteDifferences/onefactoroperator.hpp File Reference

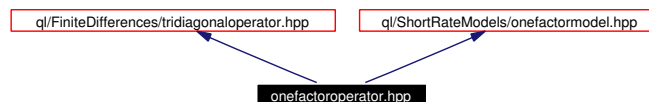
10.66.1 Detailed Description

general differential operator for one-factor interest rate models

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for onefactoroperator.hpp:



Namespaces

- namespace **QuantLib**

10.67 ql/FiniteDifferences/shoutcondition.hpp File Reference

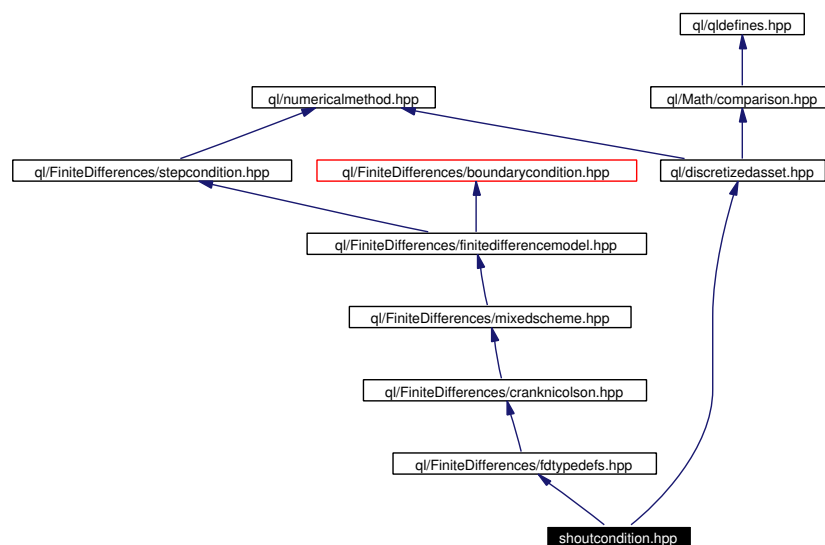
10.67.1 Detailed Description

shout option exercise condition

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for shoutcondition.hpp:



Namespaces

- namespace **QuantLib**

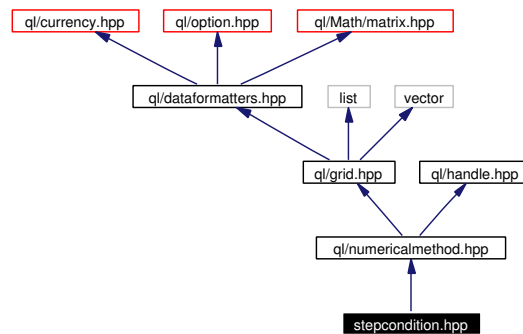
10.68 ql/FiniteDifferences/stepcondition.hpp File Reference

10.68.1 Detailed Description

conditions to be applied at every time step

```
#include <ql/numericalmethod.hpp>
```

Include dependency graph for stepcondition.hpp:



Namespaces

- namespace **QuantLib**

10.69 ql/FiniteDifferences/tridiagonaloperator.hpp File Reference

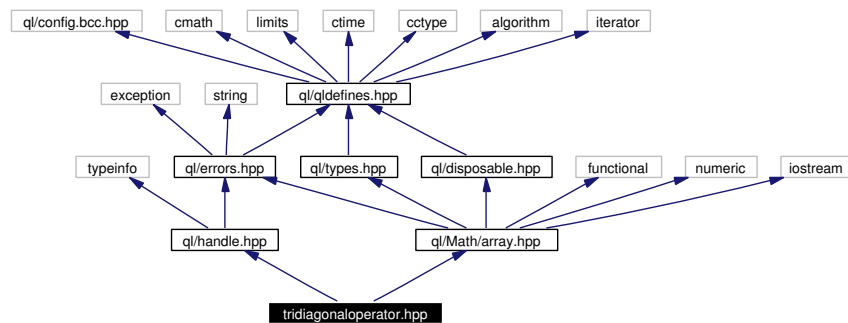
10.69.1 Detailed Description

tridiagonal operator

```
#include <ql/Math/array.hpp>
```

```
#include <ql/handle.hpp>
```

Include dependency graph for tridiagonaloperator.hpp:



Namespaces

- namespace **QuantLib**

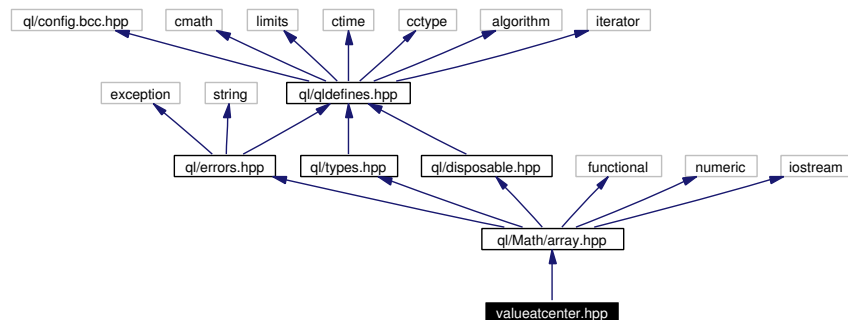
10.70 ql/FiniteDifferences/valueatcenter.hpp File Reference

10.70.1 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**

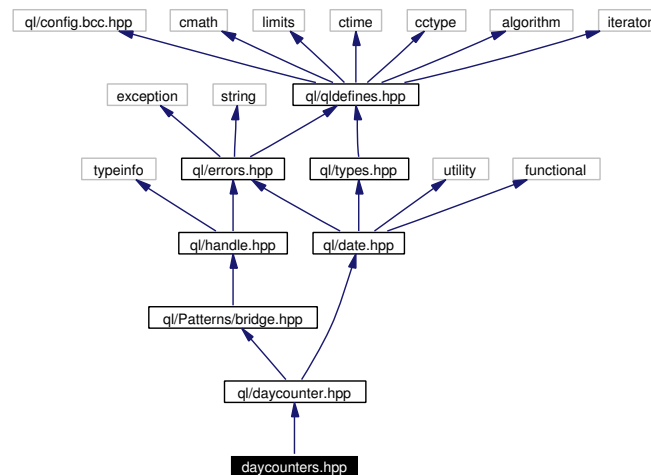
10.71 ql/functions/daycounters.hpp File Reference

10.71.1 Detailed Description

day counters functions

```
#include <ql/daycounter.hpp>
```

Include dependency graph for daycounters.hpp:



Namespaces

- namespace **QuantLib**

10.72 ql/functions/mathf.hpp File Reference

10.72.1 Detailed Description

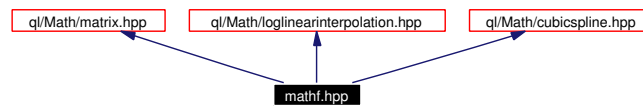
math functions

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Math/loglinearinterpolation.hpp>
```

```
#include <ql/Math/cubicspline.hpp>
```

Include dependency graph for mathf.hpp:



Namespaces

- namespace **QuantLib**

10.73 ql/functions/vols.hpp File Reference

10.73.1 Detailed Description

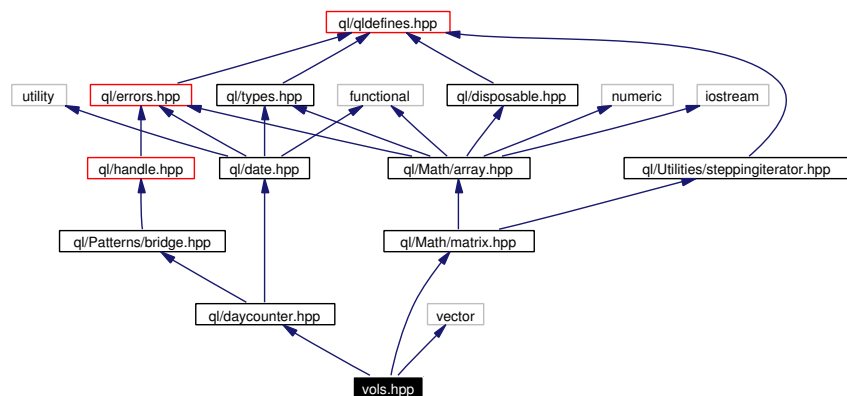
Volatility functions.

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/daycounter.hpp>
```

```
#include <vector>
```

Include dependency graph for vols.hpp:



Namespaces

- namespace **QuantLib**

10.74 ql/grid.hpp File Reference

10.74.1 Detailed Description

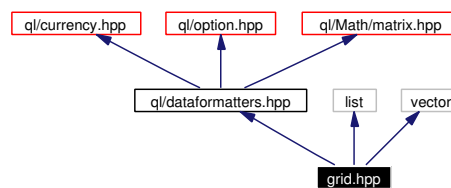
Grid classes with useful constructors for trees and finite diffs.

```
#include <ql/dataformatters.hpp>
```

```
#include <list>
```

```
#include <vector>
```

Include dependency graph for grid.hpp:



Namespaces

- namespace **QuantLib**

10.75 ql/handle.hpp File Reference

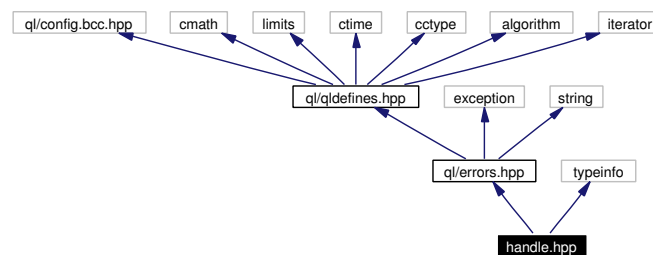
10.75.1 Detailed Description

Reference-counted pointer.

```
#include <ql/errors.hpp>
```

```
#include <typeinfo>
```

Include dependency graph for handle.hpp:



Namespaces

- namespace **QuantLib**

10.76 ql/history.hpp File Reference

10.76.1 Detailed Description

history class

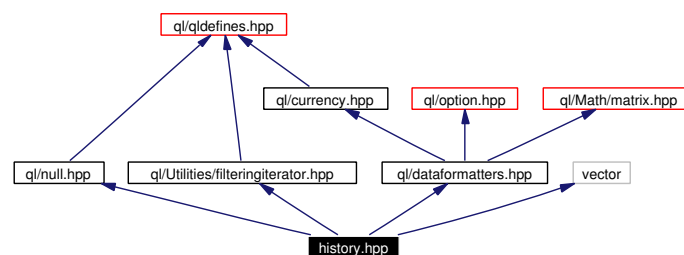
```
#include <ql/null.hpp>
```

```
#include <ql/Utilities/filteringiterator.hpp>
```

```
#include <ql/dataformatters.hpp>
```

```
#include <vector>
```

Include dependency graph for history.hpp:



Namespaces

- namespace **QuantLib**

10.77 ql/index.hpp File Reference

10.77.1 Detailed Description

purely virtual base class for indexes

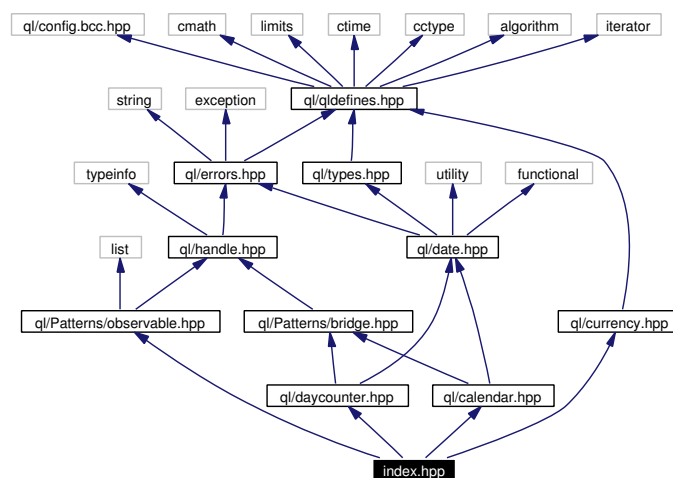
```
#include <ql/calendar.hpp>
```

```
#include <ql/currency.hpp>
```

```
#include <ql/daycounter.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for index.hpp:



Namespaces

- namespace **QuantLib**

10.78 ql/Indexes/audlibor.hpp File Reference

10.78.1 Detailed Description

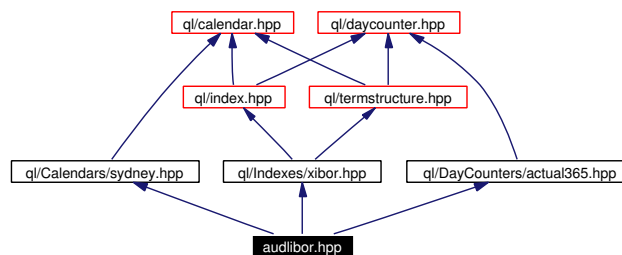
AUD Libor index (check settlement days)

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/sydney.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

Include dependency graph for audlibor.hpp:



Namespaces

- namespace **QuantLib**

10.79 ql/Indexes/cadlibor.hpp File Reference

10.79.1 Detailed Description

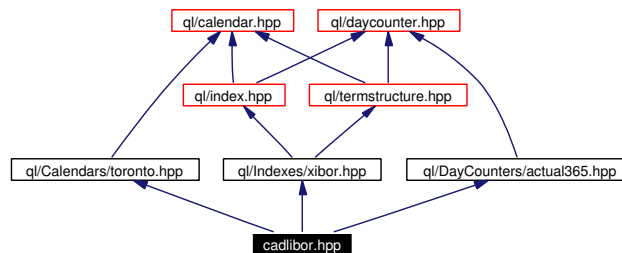
CAD Libor index (Also known as CDOR)

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/toronto.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

Include dependency graph for cadlibor.hpp:



Namespaces

- namespace **QuantLib**

10.80 ql/Indexes/chflibor.hpp File Reference

10.80.1 Detailed Description

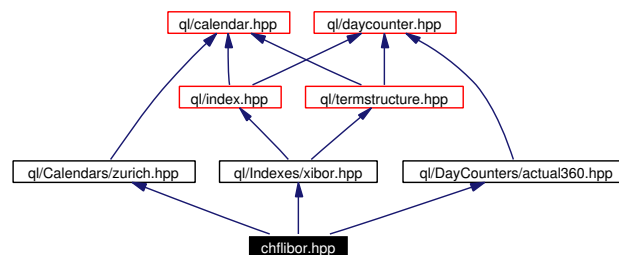
CHF Libor index (Also known as ZIBOR)

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/zurich.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

Include dependency graph for chflibor.hpp:



Namespaces

- namespace **QuantLib**

10.81 ql/Indexes/euribor.hpp File Reference

10.81.1 Detailed Description

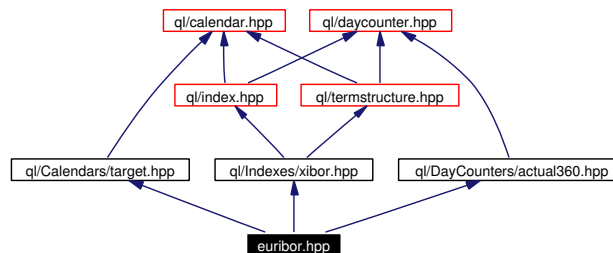
Euribor index

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

Include dependency graph for euribor.hpp:



Namespaces

- namespace **QuantLib**

10.82 ql/Indexes/gbplibor.hpp File Reference

10.82.1 Detailed Description

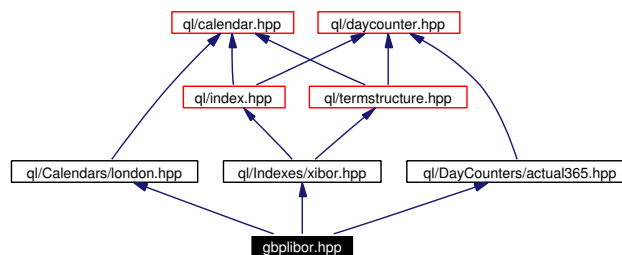
GBP Libor index

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/london.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

Include dependency graph for gbplibor.hpp:



Namespaces

- namespace **QuantLib**

10.83 ql/Indexes/jpylibor.hpp File Reference

10.83.1 Detailed Description

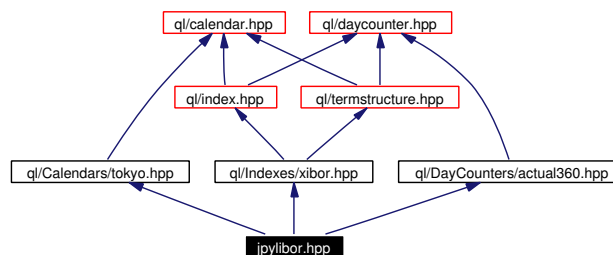
JPY Libor index (Also known as TIBOR, check settlement days)

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/tokyo.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

Include dependency graph for jpylibor.hpp:



Namespaces

- namespace **QuantLib**

10.84 ql/Indexes/usdlibor.hpp File Reference

10.84.1 Detailed Description

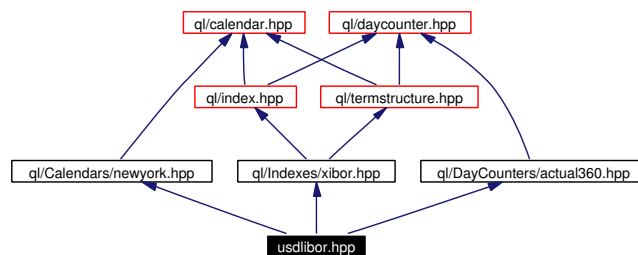
USD Libor index

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/newyork.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

Include dependency graph for usdlibor.hpp:



Namespaces

- namespace **QuantLib**

10.85 ql/Indexes/xibor.hpp File Reference

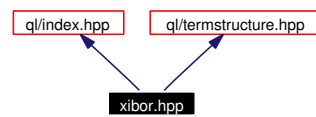
10.85.1 Detailed Description

base class for libor indexes

```
#include <ql/index.hpp>
```

```
#include <ql/termstructure.hpp>
```

Include dependency graph for xibor.hpp:



Namespaces

- namespace **QuantLib**

10.86 ql/Indexes/xibormanager.hpp File Reference

10.86.1 Detailed Description

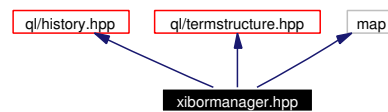
global repository for Xibor histories

```
#include <ql/history.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <map>
```

Include dependency graph for xibormanager.hpp:



Namespaces

- namespace **QuantLib**

10.87 ql/Indexes/zarlibor.hpp File Reference

10.87.1 Detailed Description

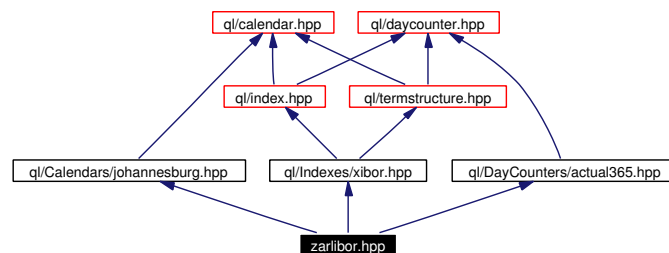
ZAR Libor index (also known as JIBAR)

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/johannesburg.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

Include dependency graph for zarlibor.hpp:



Namespaces

- namespace **QuantLib**

10.88 ql/instrument.hpp File Reference

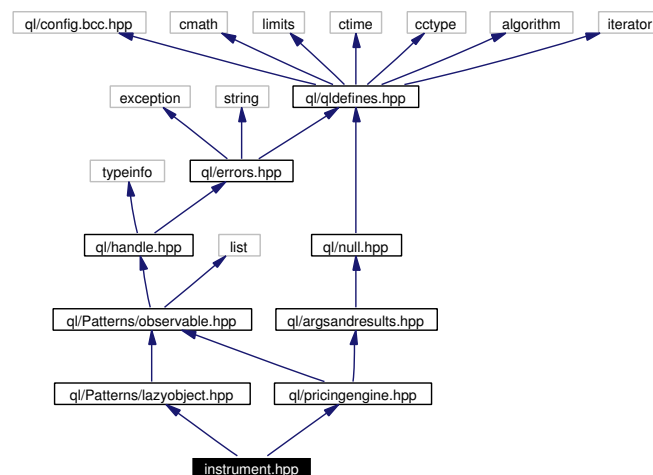
10.88.1 Detailed Description

Abstract instrument class.

```
#include <ql/Patterns/lazyobject.hpp>
```

```
#include <ql/pricingengine.hpp>
```

Include dependency graph for instrument.hpp:



Namespaces

- namespace **QuantLib**

10.89 ql/Instruments/asianoption.hpp File Reference

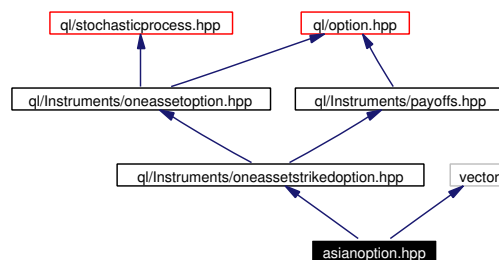
10.89.1 Detailed Description

Asian option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

```
#include <vector>
```

Include dependency graph for asianoption.hpp:



Namespaces

- namespace **QuantLib**

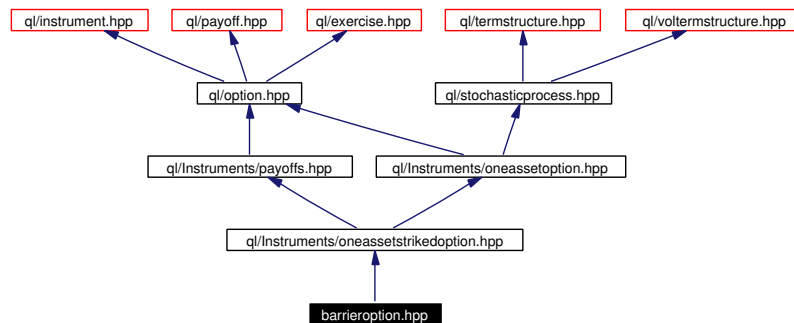
10.90 ql/Instruments/barrieroption.hpp File Reference

10.90.1 Detailed Description

Barrier option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for barrieroption.hpp:



Namespaces

- namespace **QuantLib**

10.91 ql/Instruments/basketoption.hpp File Reference

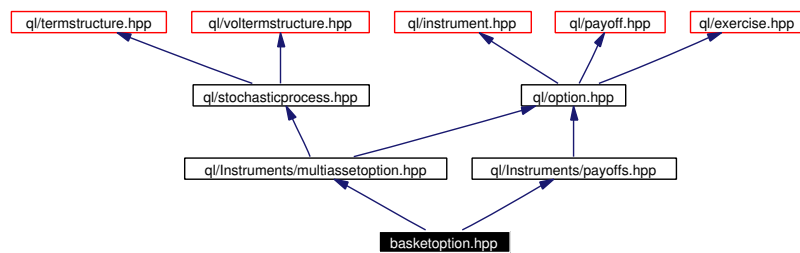
10.91.1 Detailed Description

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**

10.92 ql/Instruments/capfloor.hpp File Reference

10.92.1 Detailed Description

Cap and Floor class.

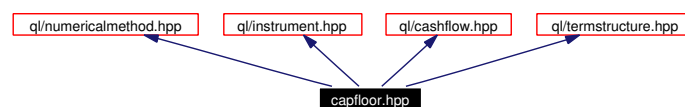
```
#include <ql/numericalmethod.hpp>
```

```
#include <ql/instrument.hpp>
```

```
#include <ql/cashflow.hpp>
```

```
#include <ql/termstructure.hpp>
```

Include dependency graph for capfloor.hpp:



Namespaces

- namespace **QuantLib**

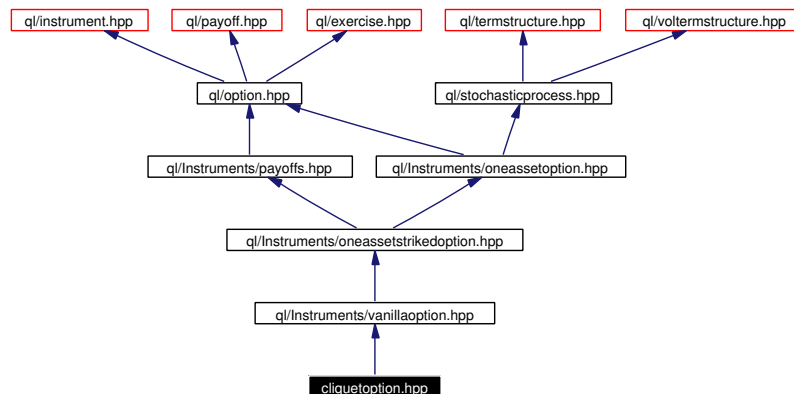
10.93 ql/Instruments/cliquestoption.hpp File Reference

10.93.1 Detailed Description

Cliquet option.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for cliquestoption.hpp:



Namespaces

- namespace **QuantLib**

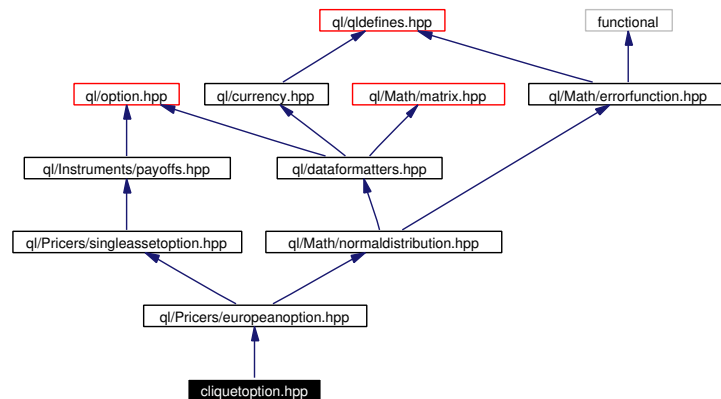
10.94 ql/Pricers/cliquetoption.hpp File Reference

10.94.1 Detailed Description

Cliquet option.

```
#include <ql/Pricers/europeanoption.hpp>
```

Include dependency graph for cliquetoption.hpp:



Namespaces

- namespace **QuantLib**

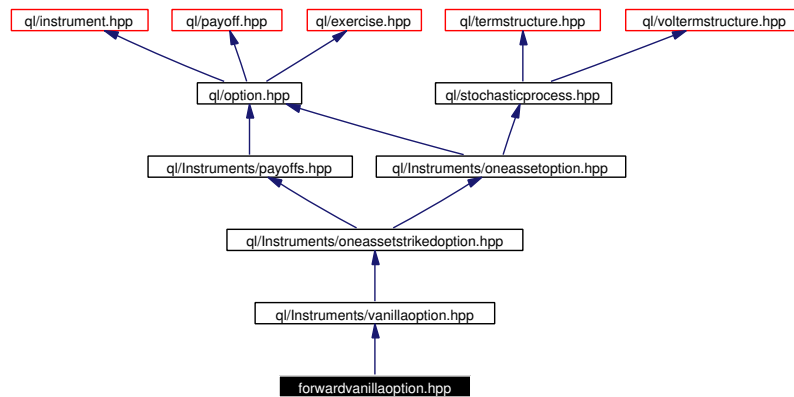
10.95 ql/Instruments/forwardvanillaoption.hpp File Reference

10.95.1 Detailed Description

Forward version of a vanilla option.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for forwardvanillaoption.hpp:



Namespaces

- namespace **QuantLib**

10.96 ql/Instruments/multiassetoption.hpp File Reference

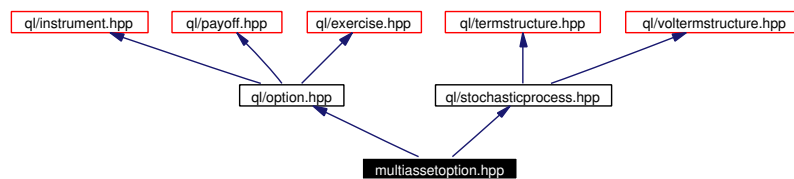
10.96.1 Detailed Description

Option on multiple assets.

```
#include <ql/option.hpp>
```

```
#include <ql/stochasticprocess.hpp>
```

Include dependency graph for multiassetoption.hpp:



Namespaces

- namespace **QuantLib**

10.97 ql/Instruments/oneassetoption.hpp File Reference

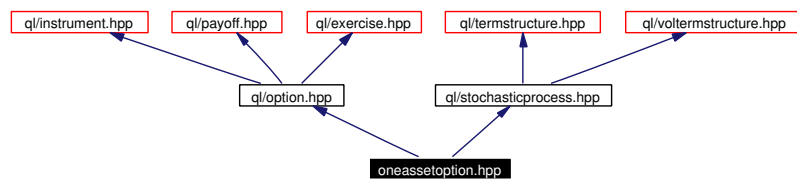
10.97.1 Detailed Description

Option on a single asset.

```
#include <ql/option.hpp>
```

```
#include <ql/stochasticprocess.hpp>
```

Include dependency graph for oneassetoption.hpp:



Namespaces

- namespace **QuantLib**

10.98 ql/Instruments/oneassetstrikedoption.hpp File Reference

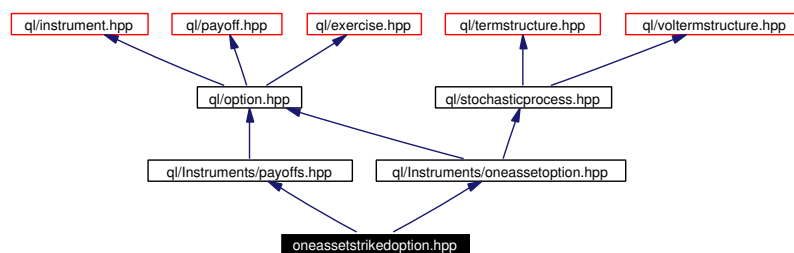
10.98.1 Detailed Description

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**

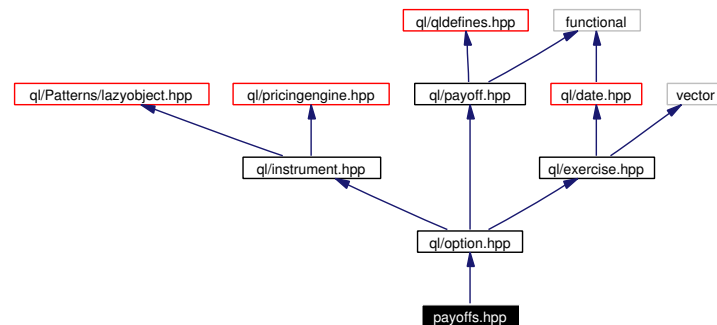
10.99 ql/Instruments/payoffs.hpp File Reference

10.99.1 Detailed Description

Payoffs for various options.

```
#include <ql/option.hpp>
```

Include dependency graph for `payoffs.hpp`:



Namespaces

- namespace **QuantLib**

10.100 ql/Instruments/quantoforwardvanillaoption.hpp File Reference

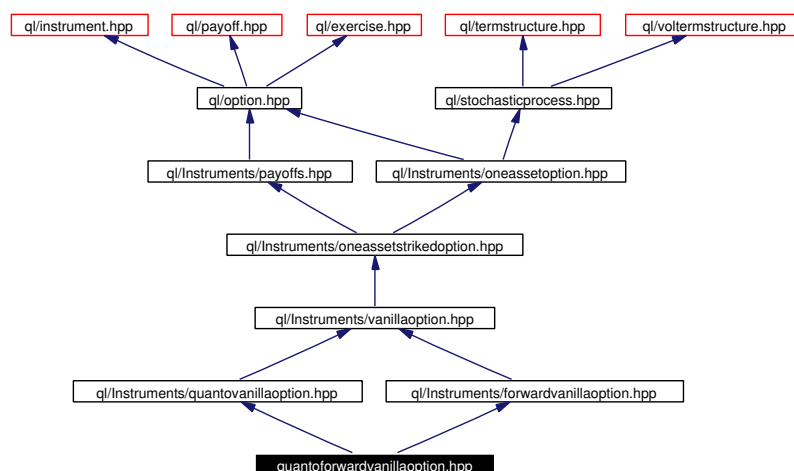
10.100.1 Detailed Description

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**

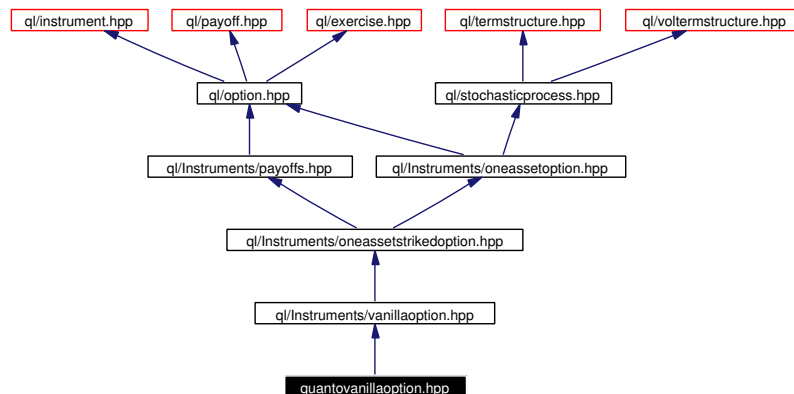
10.101 ql/Instruments/quantovanillaoption.hpp File Reference

10.101.1 Detailed Description

Quanto version of a vanilla option.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for quantovanillaoption.hpp:



Namespaces

- namespace **QuantLib**

10.102 ql/Instruments/simpleswap.hpp File Reference

10.102.1 Detailed Description

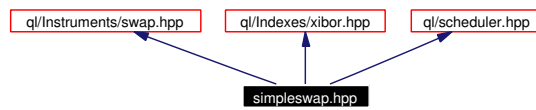
Simple fixed-rate vs Libor swap.

```
#include <ql/Instruments/swap.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/scheduler.hpp>
```

Include dependency graph for simpleswap.hpp:



Namespaces

- namespace **QuantLib**

10.103 ql/Instruments/stock.hpp File Reference

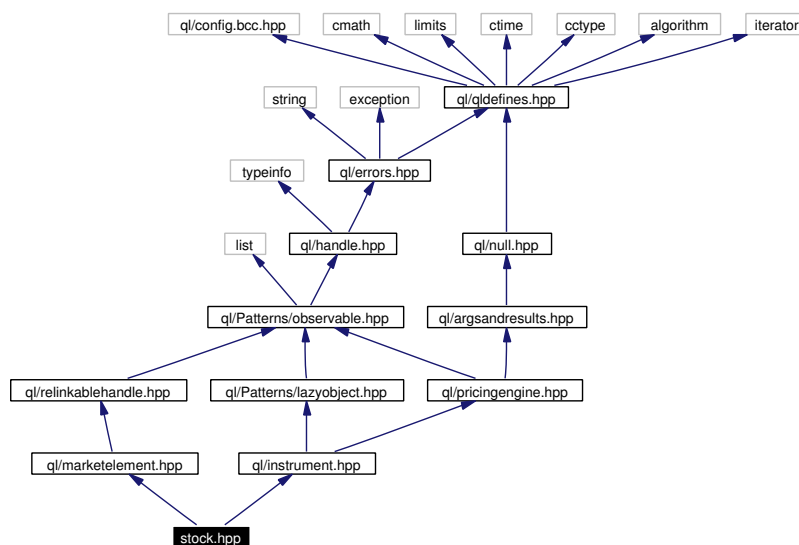
10.103.1 Detailed Description

concrete stock class

```
#include <ql/instrument.hpp>
```

```
#include <ql/marketelement.hpp>
```

Include dependency graph for stock.hpp:



Namespaces

- namespace **QuantLib**

10.104 ql/Instruments/swap.hpp File Reference

10.104.1 Detailed Description

Interest rate swap.

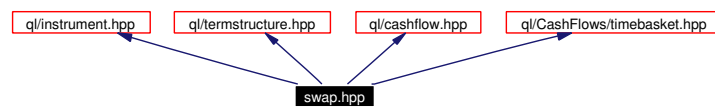
```
#include <ql/instrument.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <ql/cashflow.hpp>
```

```
#include <ql/CashFlows/timebasket.hpp>
```

Include dependency graph for swap.hpp:



Namespaces

- namespace **QuantLib**

10.105 ql/Instruments/swaption.hpp File Reference

10.105.1 Detailed Description

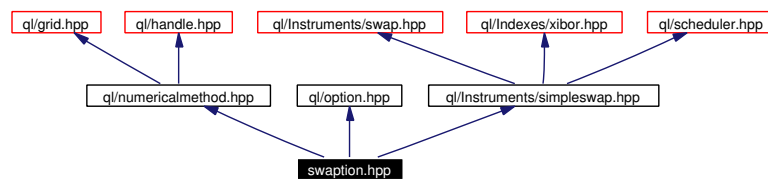
Swaption class.

```
#include <ql/numericalmethod.hpp>
```

```
#include <ql/option.hpp>
```

```
#include <ql/Instruments/simpleswap.hpp>
```

Include dependency graph for swaption.hpp:



Namespaces

- namespace **QuantLib**

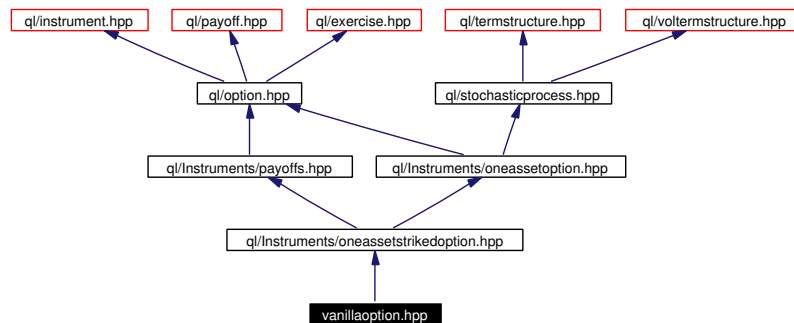
10.106 ql/Instruments/vanillaoption.hpp File Reference

10.106.1 Detailed Description

Vanilla option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for vanillaoption.hpp:



Namespaces

- namespace **QuantLib**

10.107 ql/Lattices/binomialtree.hpp File Reference

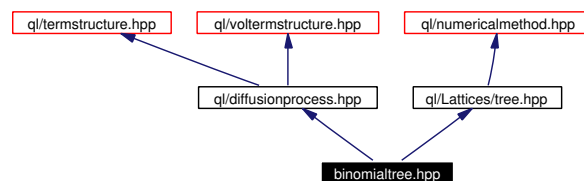
10.107.1 Detailed Description

Binomial tree class.

```
#include <ql/diffusionprocess.hpp>
```

```
#include <ql/Lattices/tree.hpp>
```

Include dependency graph for binomialtree.hpp:



Namespaces

- namespace **QuantLib**

10.108 ql/Lattices/bsmlattice.hpp File Reference

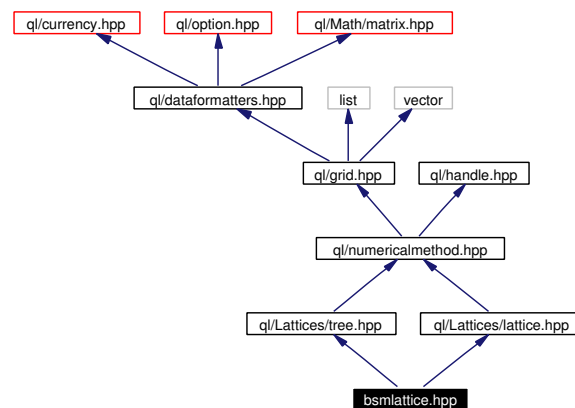
10.108.1 Detailed Description

Binomial trees under the BSM model.

```
#include <ql/Lattices/tree.hpp>
```

```
#include <ql/Lattices/lattice.hpp>
```

Include dependency graph for bsmlattice.hpp:



Namespaces

- namespace **QuantLib**

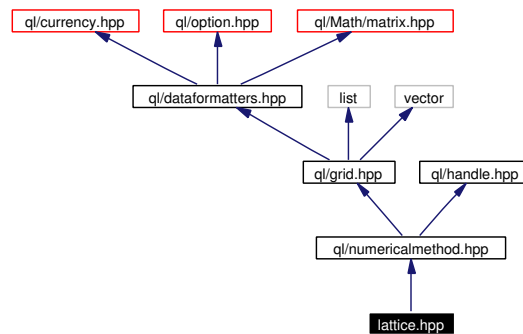
10.109 ql/Lattices/lattice.hpp File Reference

10.109.1 Detailed Description

Lattice method class.

```
#include <ql/numericalmethod.hpp>
```

Include dependency graph for lattice.hpp:



Namespaces

- namespace **QuantLib**

10.110 ql/Lattices/lattice2d.hpp File Reference

10.110.1 Detailed Description

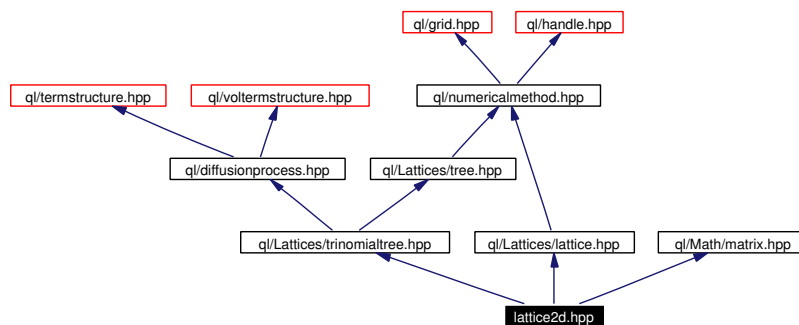
Two-dimensional tree class.

```
#include <ql/Lattices/lattice.hpp>
```

```
#include <ql/Lattices/trinomialtree.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for lattice2d.hpp:



Namespaces

- namespace **QuantLib**

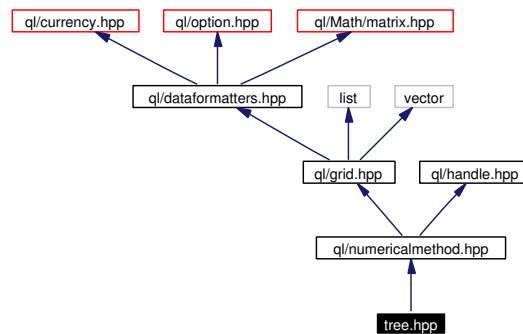
10.111 ql/Lattices/tree.hpp File Reference

10.111.1 Detailed Description

Tree class.

```
#include <ql/numericalmethod.hpp>
```

Include dependency graph for tree.hpp:



Namespaces

- namespace **QuantLib**

10.112 ql/Lattices/trinomialtree.hpp File Reference

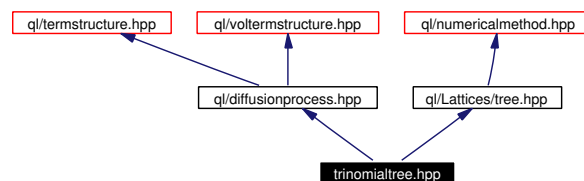
10.112.1 Detailed Description

Trinomial tree class.

```
#include <ql/diffusionprocess.hpp>
```

```
#include <ql/Lattices/tree.hpp>
```

Include dependency graph for trinomialtree.hpp:



Namespaces

- namespace **QuantLib**

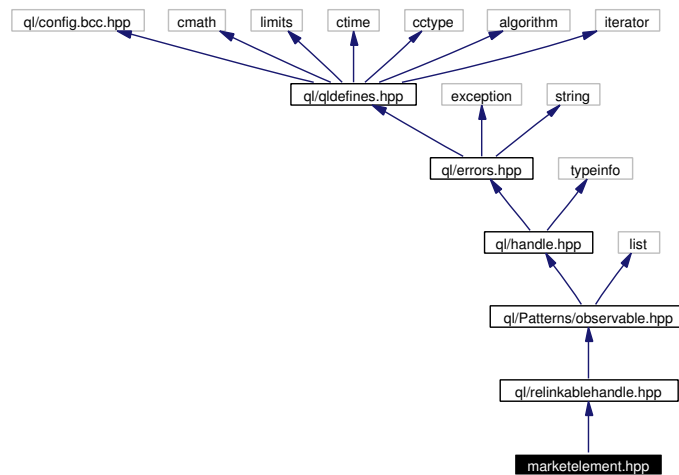
10.113 ql/marketelement.hpp File Reference

10.113.1 Detailed Description

purely virtual base class for market observables

```
#include <ql/relinkablehandle.hpp>
```

Include dependency graph for marketelement.hpp:



Namespaces

- namespace **QuantLib**

10.114 ql/Math/array.hpp File Reference

10.114.1 Detailed Description

1-D array used in linear algebra.

```
#include <ql/types.hpp>
```

```
#include <ql/errors.hpp>
```

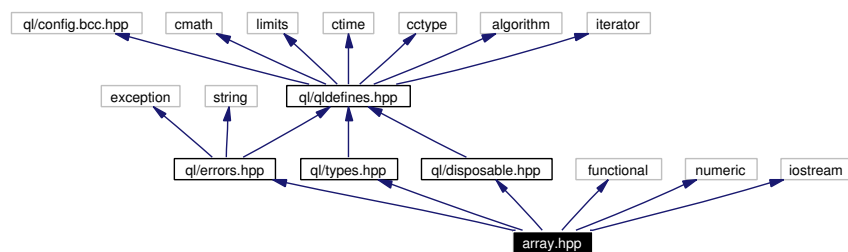
```
#include <ql/disposable.hpp>
```

```
#include <functional>
```

```
#include <numeric>
```

```
#include <iostream>
```

Include dependency graph for array.hpp:



Namespaces

- namespace **QuantLib**

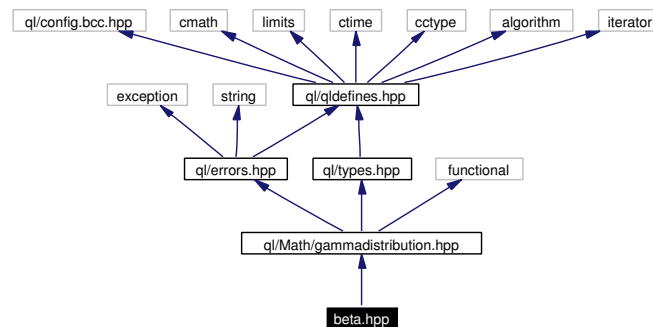
10.115 ql/Math/beta.hpp File Reference

10.115.1 Detailed Description

Beta and beta incomplete functions.

```
#include <ql/Math/gammadistribution.hpp>
```

Include dependency graph for beta.hpp:



Namespaces

- namespace **QuantLib**

10.116 ql/Math/bicubicsplineinterpolation.hpp File Reference

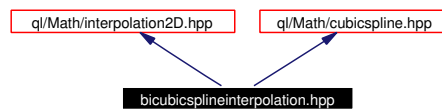
10.116.1 Detailed Description

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**

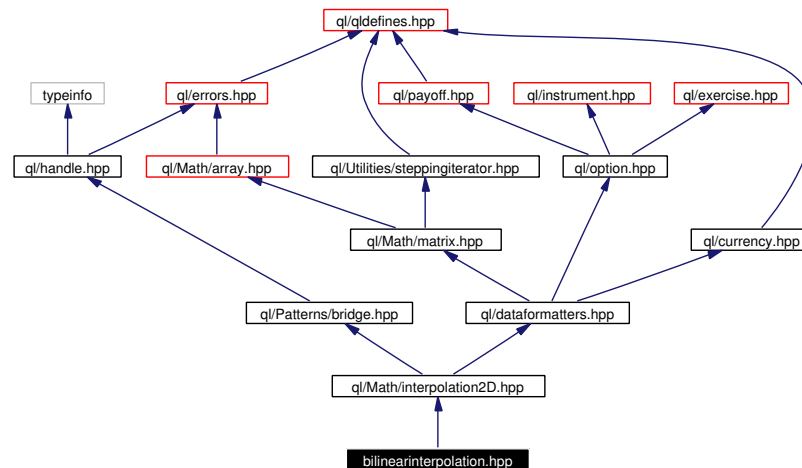
10.117 ql/Math/bilinearinterpolation.hpp File Reference

10.117.1 Detailed Description

bilinear interpolation between discrete points

```
#include <ql/Math/interpolation2D.hpp>
```

Include dependency graph for bilinearinterpolation.hpp:



Namespaces

- namespace **QuantLib**

10.118 ql/Math/binomialdistribution.hpp File Reference

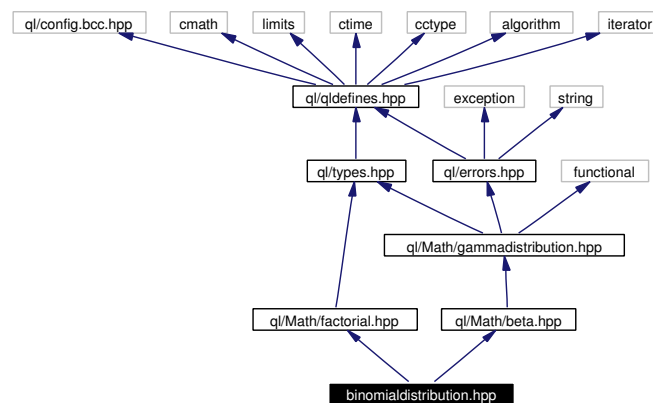
10.118.1 Detailed Description

Binomial distribution.

```
#include <ql/Math/factorial.hpp>
```

```
#include <ql/Math/beta.hpp>
```

Include dependency graph for binomialdistribution.hpp:



Namespaces

- namespace **QuantLib**

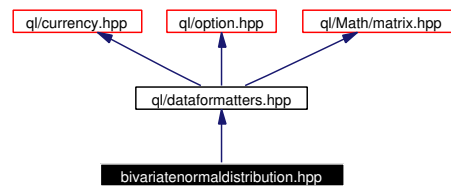
10.119 ql/Math/bivariatenormaldistribution.hpp File Reference

10.119.1 Detailed Description

bivariate cumulative normal distribution

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for bivariatenormaldistribution.hpp:



Namespaces

- namespace **QuantLib**

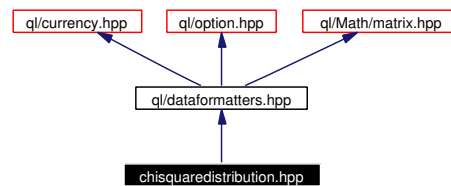
10.120 ql/Math/chisquaredistribution.hpp File Reference

10.120.1 Detailed Description

Chi-square (central and non-central) distributions.

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for chisquaredistribution.hpp:



Namespaces

- namespace **QuantLib**

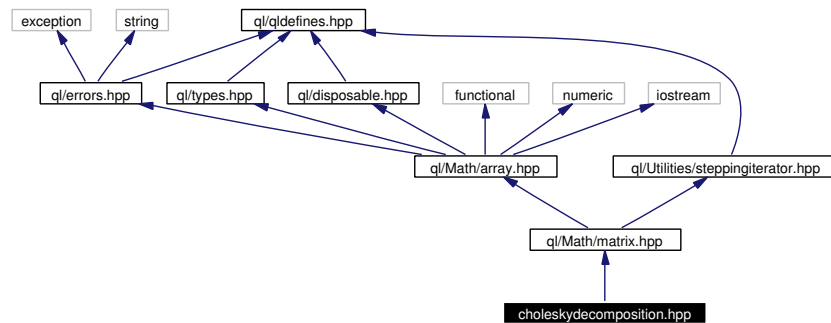
10.121 ql/Math/choleskydecomposition.hpp File Reference

10.121.1 Detailed Description

Cholesky decomposition.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for choleskydecomposition.hpp:



Namespaces

- namespace **QuantLib**

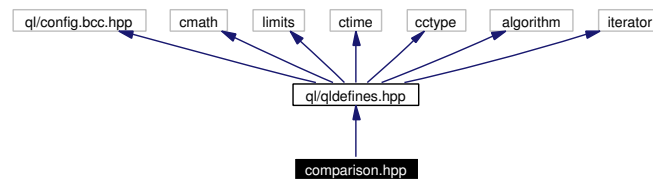
10.122 ql/Math/comparison.hpp File Reference

10.122.1 Detailed Description

floating-point comparisons

#include <ql/qldefines.hpp>

Include dependency graph for comparison.hpp:



Namespaces

- namespace **QuantLib**

10.123 ql/Math/cubicspline.hpp File Reference

10.123.1 Detailed Description

cubic spline interpolation between discrete points

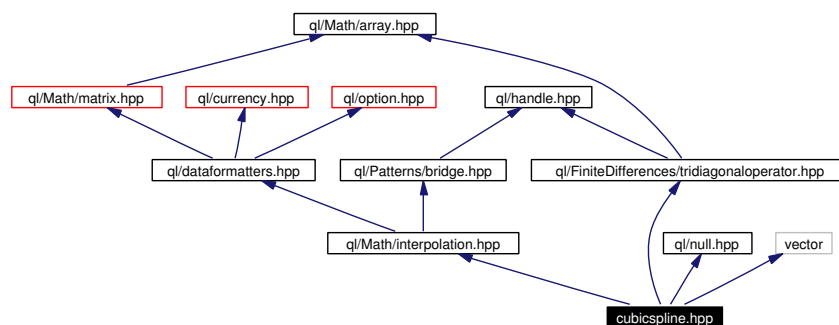
```
#include <ql/Math/interpolation.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/null.hpp>
```

```
#include <vector>
```

Include dependency graph for cubicspline.hpp:



Namespaces

- namespace **QuantLib**

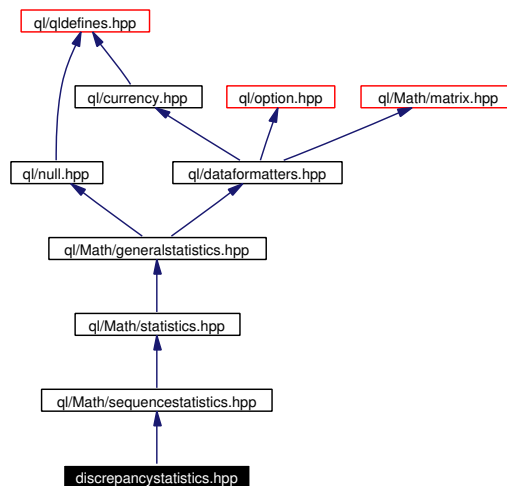
10.124 ql/Math/discrepancystatistics.hpp File Reference

10.124.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

```
#include <ql/Math/sequencestatistics.hpp>
```

Include dependency graph for discrepandystatistics.hpp:



Namespaces

- namespace **QuantLib**

10.125 ql/Math/errorfunction.hpp File Reference

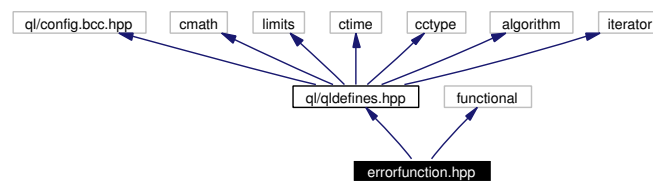
10.125.1 Detailed Description

Error function.

```
#include <ql/qldefines.hpp>
```

```
#include <functional>
```

Include dependency graph for errorfunction.hpp:



Namespaces

- namespace **QuantLib**

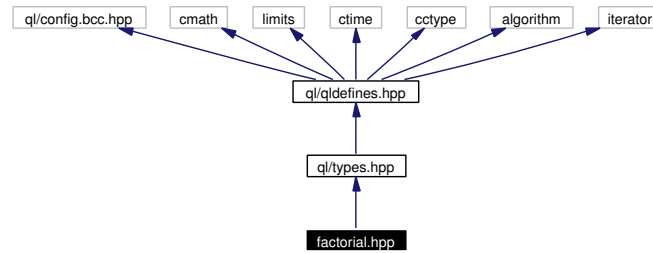
10.126 ql/Math/factorial.hpp File Reference

10.126.1 Detailed Description

Factorial numbers calculator.

```
#include <ql/types.hpp>
```

Include dependency graph for factorial.hpp:



Namespaces

- namespace **QuantLib**

10.127 ql/Math/functional.hpp File Reference

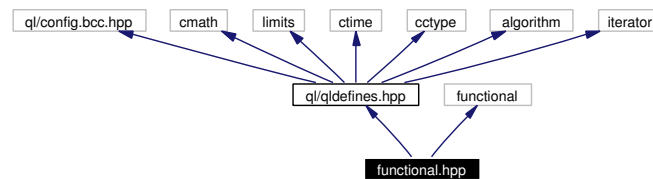
10.127.1 Detailed Description

functionals and combinators not included in the STL

```
#include <ql/qldefines.hpp>
```

```
#include <functional>
```

Include dependency graph for functional.hpp:



Namespaces

- namespace **QuantLib**

10.128 ql/Math/gammadistribution.hpp File Reference

10.128.1 Detailed Description

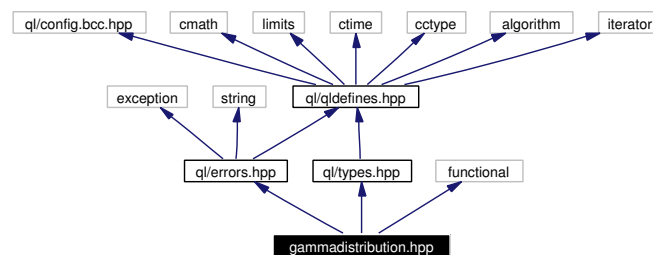
Gamma distribution.

```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for gammadistribution.hpp:



Namespaces

- namespace **QuantLib**

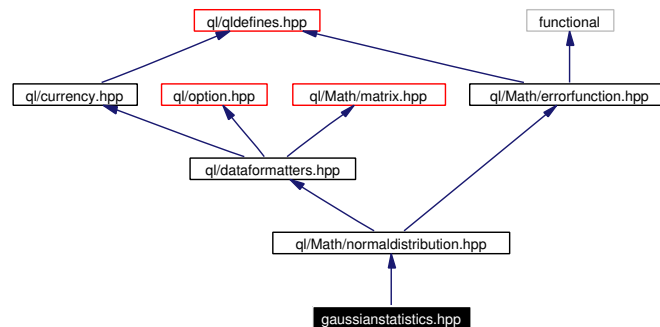
10.129 ql/Math/gaussianstatistics.hpp File Reference

10.129.1 Detailed Description

statistics tool for gaussian-assumption risk measures

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for gaussianstatistics.hpp:



Namespaces

- namespace **QuantLib**

10.130 ql/Math/generalstatistics.hpp File Reference

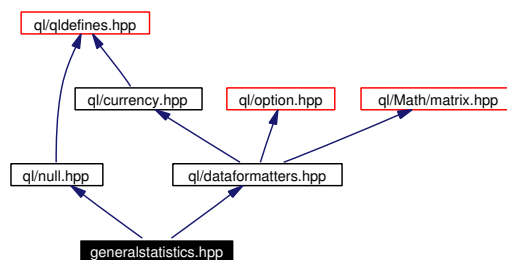
10.130.1 Detailed Description

statistics tool

```
#include <ql/null.hpp>
```

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for generalstatistics.hpp:



Namespaces

- namespace **QuantLib**

10.131 ql/Math/incompletegamma.hpp File Reference

10.131.1 Detailed Description

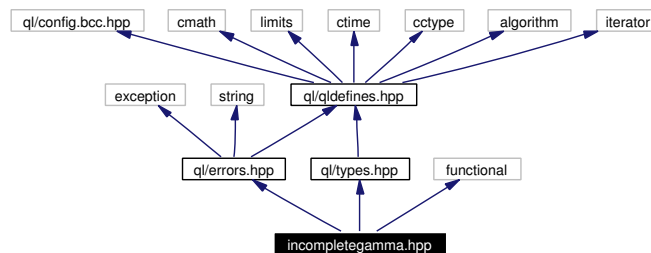
Incomplete Gamma function.

```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for incompletegamma.hpp:



Namespaces

- namespace **QuantLib**

10.132 ql/Math/incrementalstatistics.hpp File Reference

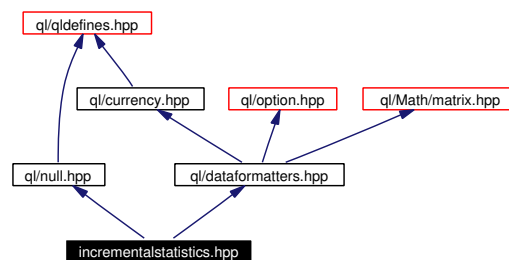
10.132.1 Detailed Description

statistics tool based on incremental accumulation

```
#include <ql/null.hpp>
```

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for incrementalstatistics.hpp:



Namespaces

- namespace **QuantLib**

10.133 ql/Math/interpolation.hpp File Reference

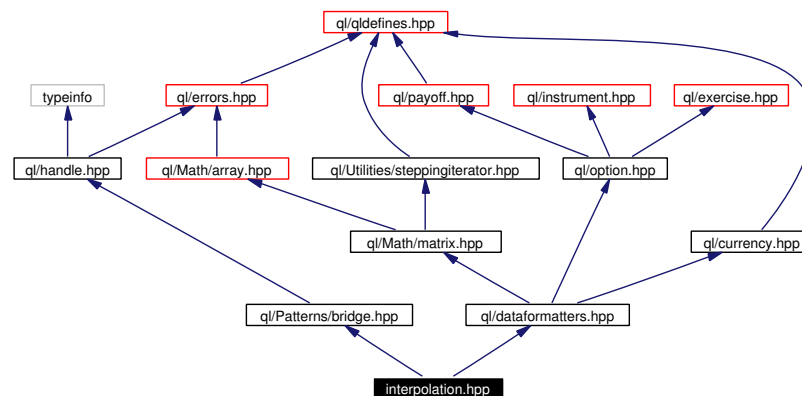
10.133.1 Detailed Description

base class for 1-D interpolations

```
#include <ql/Patterns/bridge.hpp>
```

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for interpolation.hpp:



Namespaces

- namespace **QuantLib**

10.134 ql/Math/interpolation2D.hpp File Reference

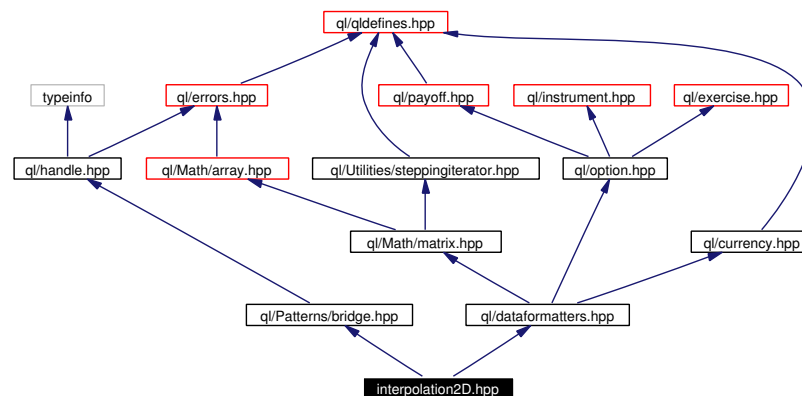
10.134.1 Detailed Description

abstract base classes for 2-D interpolations

```
#include <ql/Patterns/bridge.hpp>
```

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for interpolation2D.hpp:



Namespaces

- namespace **QuantLib**

10.135 ql/Math/interpolationtraits.hpp File Reference

10.135.1 Detailed Description

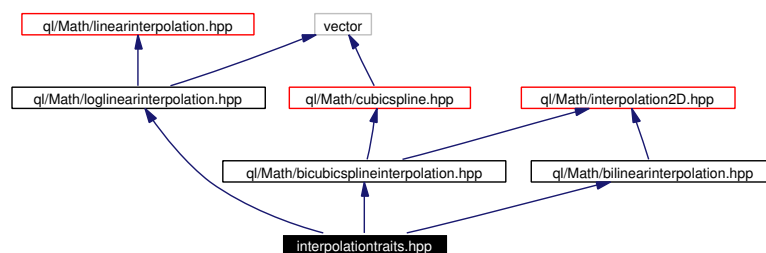
traits classes for interpolation algorithms

```
#include <ql/Math/loglinearinterpolation.hpp>
```

```
#include <ql/Math/bilinearinterpolation.hpp>
```

```
#include <ql/Math/bicubicsplineinterpolation.hpp>
```

Include dependency graph for interpolationtraits.hpp:



Namespaces

- namespace **QuantLib**

10.136 ql/Math/kronrodintegral.hpp File Reference

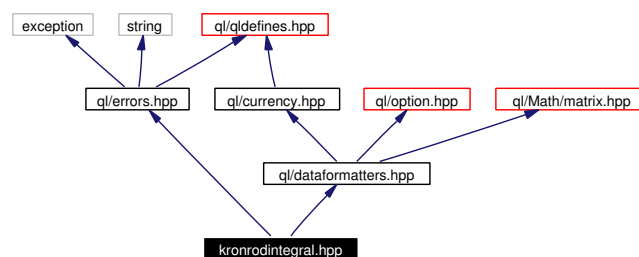
10.136.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

```
#include <ql/errors.hpp>
```

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for kronrodintegral.hpp:



Namespaces

- namespace **QuantLib**

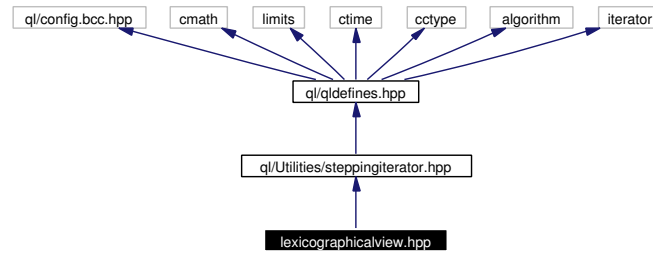
10.137 ql/Math/lexicographicalview.hpp File Reference

10.137.1 Detailed Description

Lexicographical 2-D view of a contiguous set of data.

```
#include <ql/Utilities/steppingiterator.hpp>
```

Include dependency graph for lexicographicalview.hpp:



Namespaces

- namespace **QuantLib**

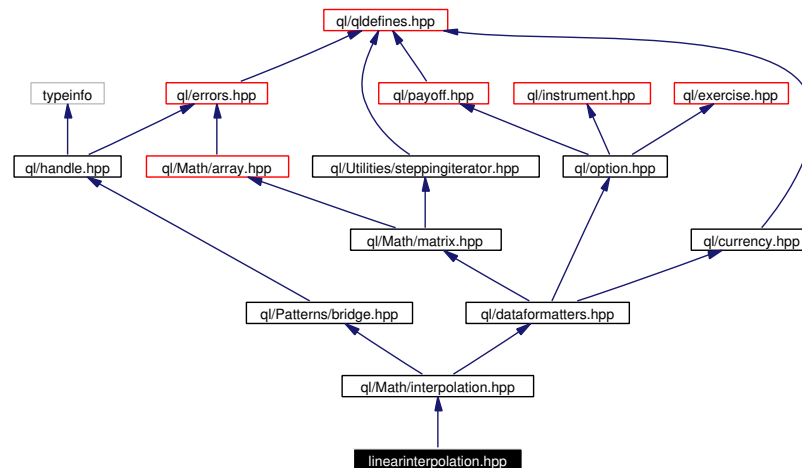
10.138 ql/Math/linearinterpolation.hpp File Reference

10.138.1 Detailed Description

linear interpolation between discrete points

```
#include <ql/Math/interpolation.hpp>
```

Include dependency graph for linearinterpolation.hpp:



Namespaces

- namespace **QuantLib**

10.139 ql/Math/loglinearinterpolation.hpp File Reference

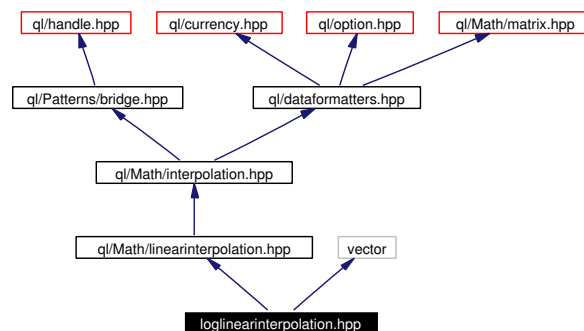
10.139.1 Detailed Description

log-linear interpolation between discrete points

```
#include <ql/Math/linearinterpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for loglinearinterpolation.hpp:



Namespaces

- namespace **QuantLib**

10.140 ql/Math/matrix.hpp File Reference

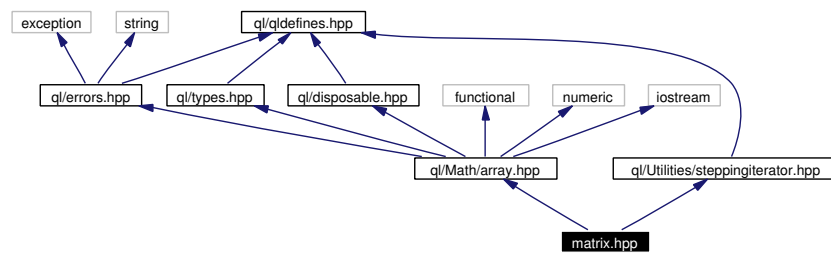
10.140.1 Detailed Description

matrix used in linear algebra.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Utilities/steppingiterator.hpp>
```

Include dependency graph for matrix.hpp:



Namespaces

- namespace **QuantLib**

10.141 ql/Math/normaldistribution.hpp File Reference

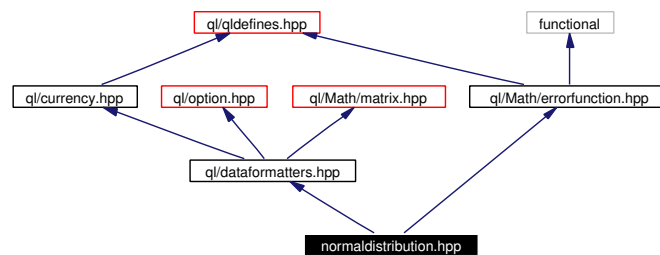
10.141.1 Detailed Description

normal, cumulative and inverse cumulative distributions

```
#include <ql/dataformatters.hpp>
```

```
#include <ql/Math/errorfunction.hpp>
```

Include dependency graph for normaldistribution.hpp:



Namespaces

- namespace **QuantLib**

10.142 ql/Math/poissondistribution.hpp File Reference

10.142.1 Detailed Description

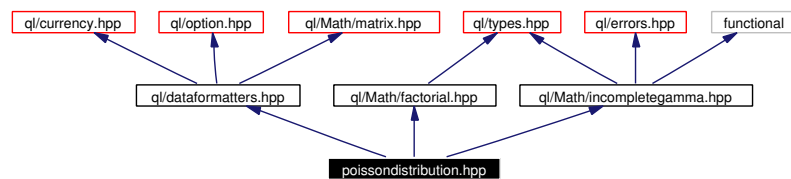
Poisson distribution.

```
#include <ql/dataformatters.hpp>
```

```
#include <ql/Math/factorial.hpp>
```

```
#include <ql/Math/incompletegamma.hpp>
```

Include dependency graph for poissondistribution.hpp:



Namespaces

- namespace **QuantLib**

10.143 ql/Math/primenumbers.hpp File Reference

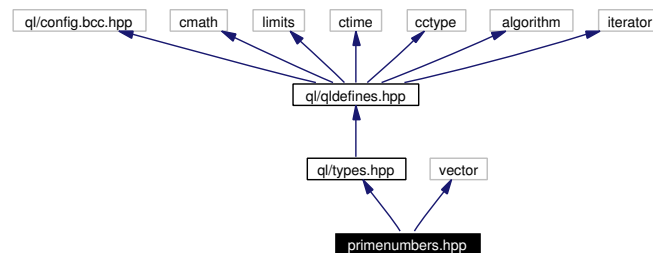
10.143.1 Detailed Description

Prime numbers calculator.

```
#include <ql/types.hpp>
```

```
#include <vector>
```

Include dependency graph for primenumbers.hpp:



Namespaces

- namespace **QuantLib**

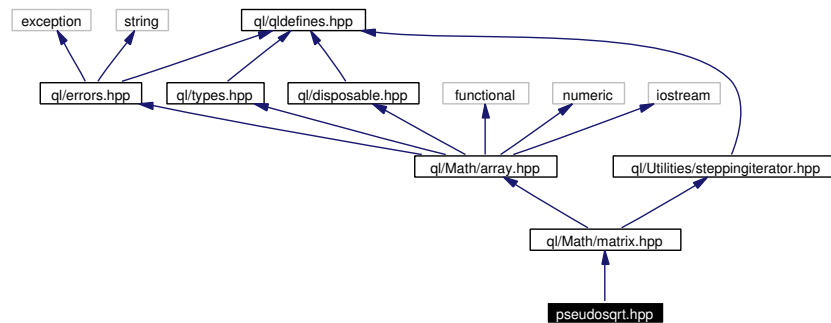
10.144 ql/Math/pseudosqrt.hpp File Reference

10.144.1 Detailed Description

pseudo square root of a real symmetric matrix

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for pseudosqrt.hpp:



Namespaces

- namespace **QuantLib**

10.145 ql/Math/riskstatistics.hpp File Reference

10.145.1 Detailed Description

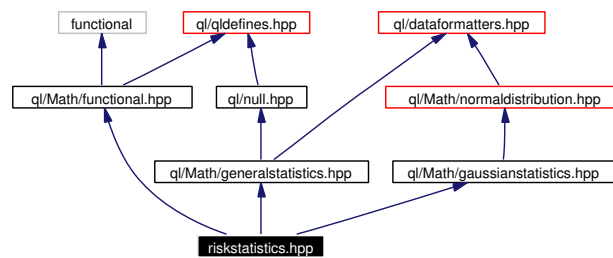
empirical-distribution risk measures

```
#include <ql/Math/functional.hpp>
```

```
#include <ql/Math/generalstatistics.hpp>
```

```
#include <ql/Math/gaussianstatistics.hpp>
```

Include dependency graph for riskstatistics.hpp:



Namespaces

- namespace **QuantLib**

10.146 ql/Math/segmentintegral.hpp File Reference

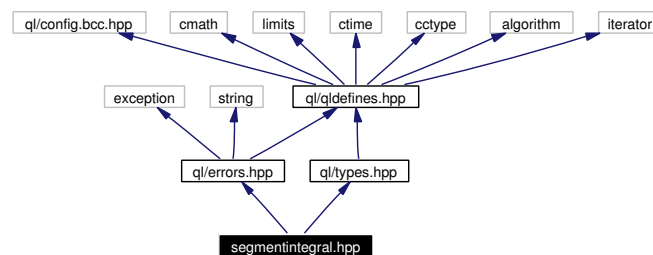
10.146.1 Detailed Description

Integral of a one-dimensional function.

```
#include <ql/types.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for segmentintegral.hpp:



Namespaces

- namespace **QuantLib**

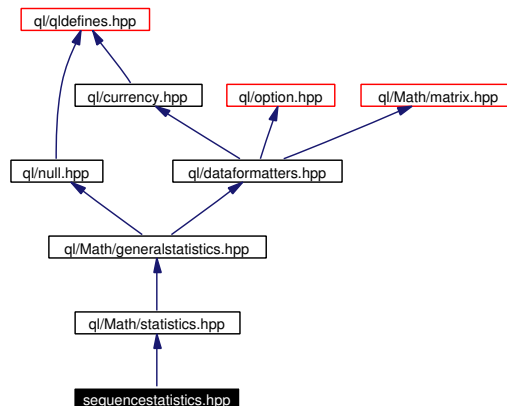
10.147 ql/Math/sequencestatistics.hpp File Reference

10.147.1 Detailed Description

Statistics tools for sequence (vector, list, array) samples.

```
#include <ql/Math/statistics.hpp>
```

Include dependency graph for sequencestatistics.hpp:



Namespaces

- namespace QuantLib

Defines

- #define DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID(METHOD)
- #define DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)

10.147.2 Define Documentation

10.147.2.1 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID(METHOD)

Value:

```
template <class Stat> \
    std::vector<double> \
    SequenceStatistics<Stat>::METHOD() const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(); \
        return results_; \
    }
```

10.147.2.2 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)

Value:

```
template <class Stat> \
    std::vector<double> \
    SequenceStatistics<Stat>::METHOD(double x) const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(x); \
        return results_; \
    }
```

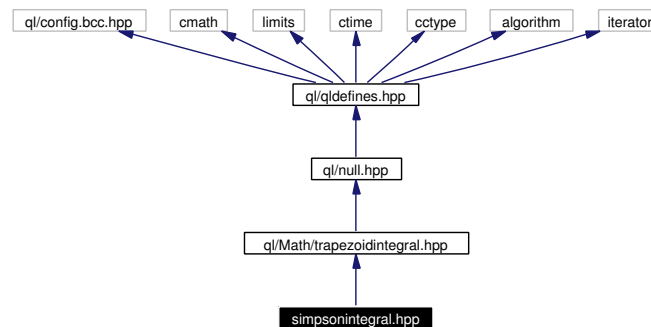
10.148 ql/Math/simpsonintegral.hpp File Reference

10.148.1 Detailed Description

integral of a one-dimensional function

```
#include <ql/Math/trapezoidintegral.hpp>
```

Include dependency graph for `simpsonintegral.hpp`:



Namespaces

- namespace **QuantLib**

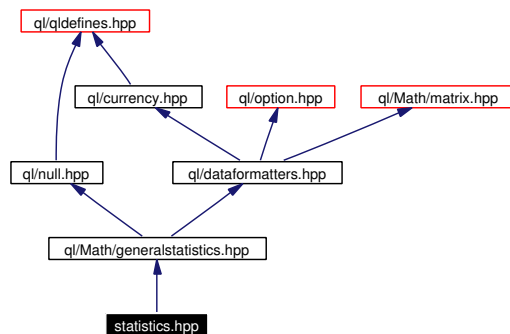
10.149 ql/Math/statistics.hpp File Reference

10.149.1 Detailed Description

statistics tool with risk measures

```
#include <ql/Math/generalstatistics.hpp>
```

Include dependency graph for statistics.hpp:



Namespaces

- namespace **QuantLib**

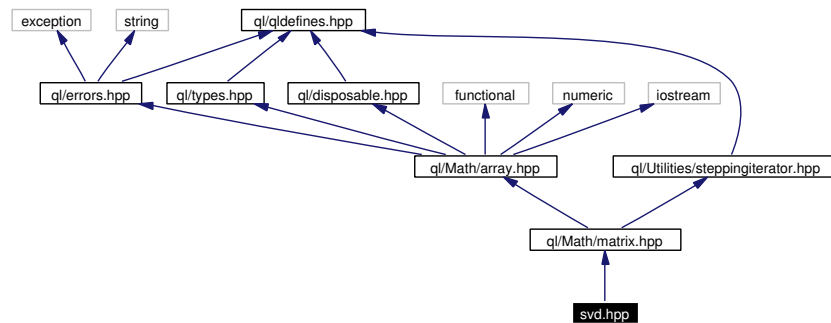
10.150 ql/Math/svd.hpp File Reference

10.150.1 Detailed Description

singular value decomposition

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for svd.hpp:



Namespaces

- namespace **QuantLib**

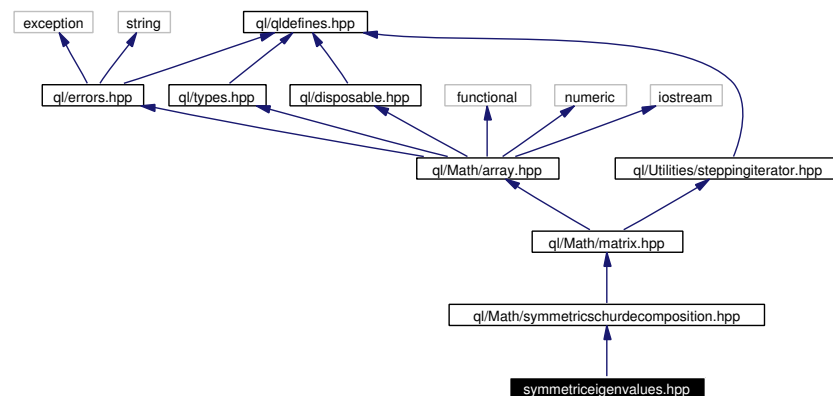
10.151 ql/Math/symmetriceigenvalues.hpp File Reference

10.151.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

```
#include <ql/Math/symmetricschurdecomposition.hpp>
```

Include dependency graph for symmetriceigenvalues.hpp:



Namespaces

- namespace **QuantLib**

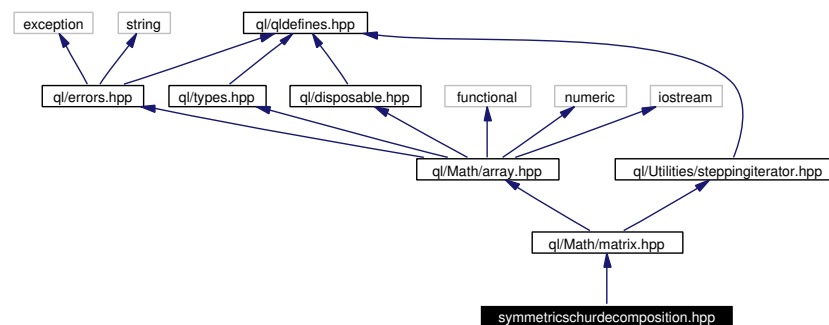
10.152 ql/Math/symmetricschurdecomposition.hpp File Reference

10.152.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for symmetricschurdecomposition.hpp:



Namespaces

- namespace **QuantLib**

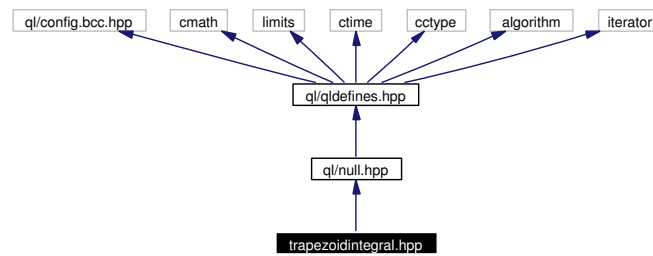
10.153 ql/Math/trapezoidintegral.hpp File Reference

10.153.1 Detailed Description

integral of a one-dimensional function

```
#include <ql/null.hpp>
```

Include dependency graph for trapezoidintegral.hpp:



Namespaces

- namespace **QuantLib**

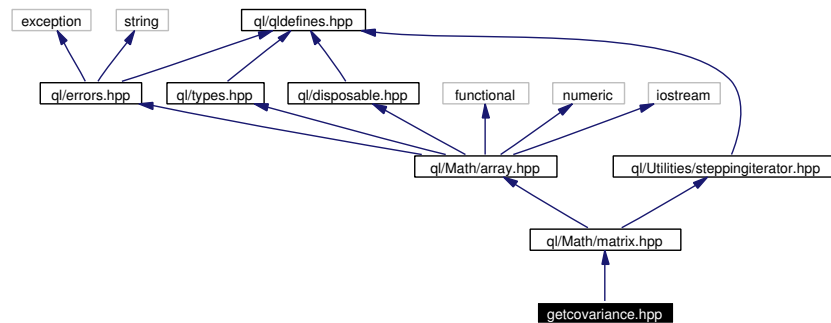
10.155 ql/MonteCarlo/getcovariance.hpp File Reference

10.155.1 Detailed Description

Covariance matrix calculation.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for getcovariance.hpp:



Namespaces

- namespace **QuantLib**

10.156 ql/MonteCarlo/mctrails.hpp File Reference

10.156.1 Detailed Description

Monte Carlo policies.

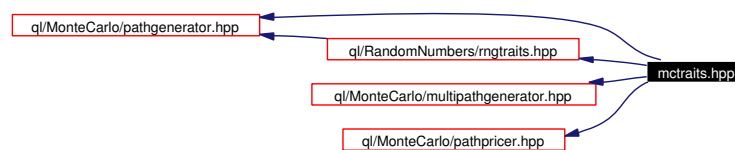
```
#include <ql/MonteCarlo/pathgenerator.hpp>
```

```
#include <ql/MonteCarlo/multipathgenerator.hpp>
```

```
#include <ql/MonteCarlo/pathpricer.hpp>
```

```
#include <ql/RandomNumbers/rngtraits.hpp>
```

Include dependency graph for mctrails.hpp:



Namespaces

- namespace **QuantLib**

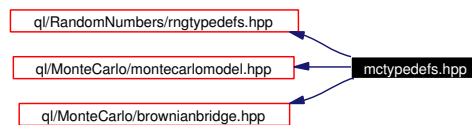
10.157 ql/MonteCarlo/mctypedefs.hpp File Reference

10.157.1 Detailed Description

Default choices for template instantiations.

```
#include <ql/RandomNumbers/rngtypedefs.hpp>
#include <ql/MonteCarlo/montecarlomodel.hpp>
#include <ql/MonteCarlo/brownianbridge.hpp>
```

Include dependency graph for mctypedefs.hpp:



Namespaces

- namespace **QuantLib**

10.158 ql/MonteCarlo/montecarlomodel.hpp File Reference

10.158.1 Detailed Description

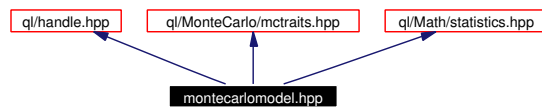
General purpose Monte Carlo model.

```
#include <ql/handle.hpp>
```

```
#include <ql/MonteCarlo/mctraits.hpp>
```

```
#include <ql/Math/statistics.hpp>
```

Include dependency graph for montecarlomodel.hpp:



Namespaces

- namespace **QuantLib**

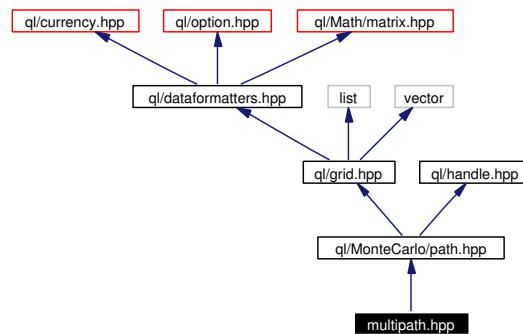
10.159 ql/MonteCarlo/multipath.hpp File Reference

10.159.1 Detailed Description

Correlated multiple asset paths.

```
#include <ql/MonteCarlo/path.hpp>
```

Include dependency graph for multipath.hpp:



Namespaces

- namespace **QuantLib**

10.160 ql/MonteCarlo/multipathgenerator.hpp File Reference

10.160.1 Detailed Description

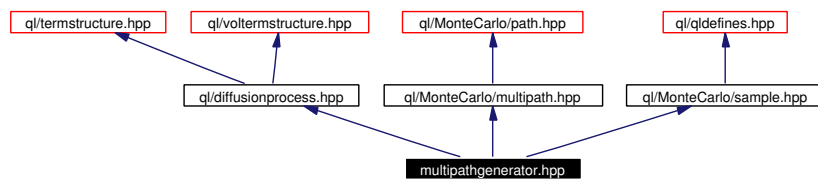
Generates a multi path from a random-array generator.

```
#include <ql/diffusionprocess.hpp>
```

```
#include <ql/MonteCarlo/multipath.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for multipathgenerator.hpp:



Namespaces

- namespace **QuantLib**

10.161 ql/MonteCarlo/path.hpp File Reference

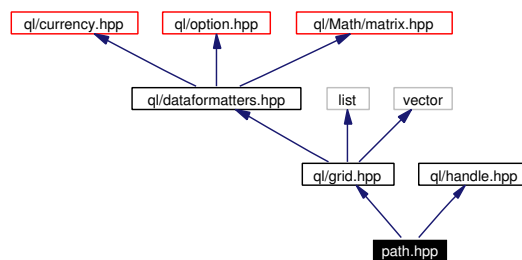
10.161.1 Detailed Description

single factor random walk

```
#include <ql/grid.hpp>
```

```
#include <ql/handle.hpp>
```

Include dependency graph for path.hpp:



Namespaces

- namespace **QuantLib**

10.162 ql/MonteCarlo/pathgenerator.hpp File Reference

10.162.1 Detailed Description

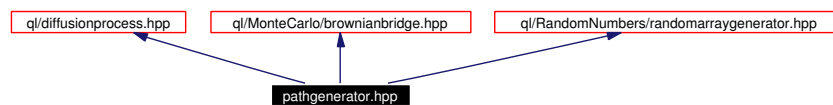
Generates random paths using a sequence generator.

```
#include <ql/diffusionprocess.hpp>
```

```
#include <ql/MonteCarlo/brownianbridge.hpp>
```

```
#include <ql/RandomNumbers/randomarraygenerator.hpp>
```

Include dependency graph for pathgenerator.hpp:



Namespaces

- namespace **QuantLib**

10.163 ql/MonteCarlo/pathpricer.hpp File Reference

10.163.1 Detailed Description

base class for single-path pricers

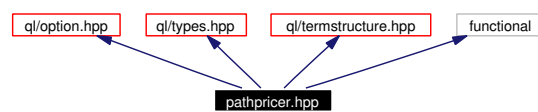
```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <functional>
```

Include dependency graph for pathpricer.hpp:



Namespaces

- namespace **QuantLib**

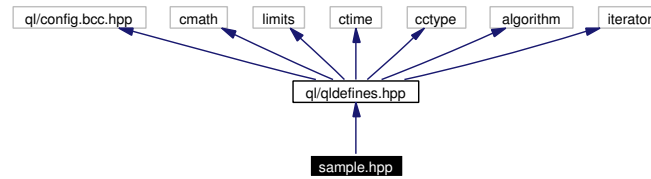
10.164 ql/MonteCarlo/sample.hpp File Reference

10.164.1 Detailed Description

weighted sample

```
#include <ql/qldefines.hpp>
```

Include dependency graph for sample.hpp:



Namespaces

- namespace **QuantLib**

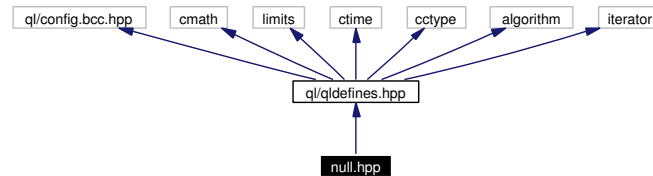
10.165 ql/null.hpp File Reference

10.165.1 Detailed Description

null values

```
#include <ql/qldefines.hpp>
```

Include dependency graph for null.hpp:



Namespaces

- namespace **QuantLib**

10.166 ql/numericalmethod.hpp File Reference

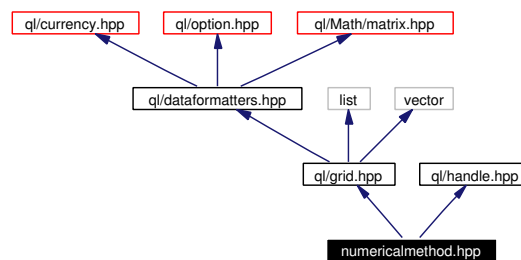
10.166.1 Detailed Description

Numerical method class.

```
#include <ql/grid.hpp>
```

```
#include <ql/handle.hpp>
```

Include dependency graph for numericalmethod.hpp:



Namespaces

- namespace **QuantLib**

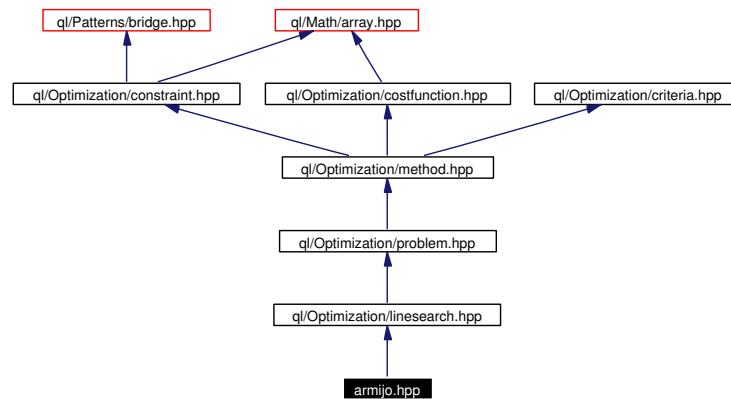
10.167 ql/Optimization/armijo.hpp File Reference

10.167.1 Detailed Description

Armijo line-search class.

```
#include <ql/Optimization/linesearch.hpp>
```

Include dependency graph for armijo.hpp:



Namespaces

- namespace **QuantLib**

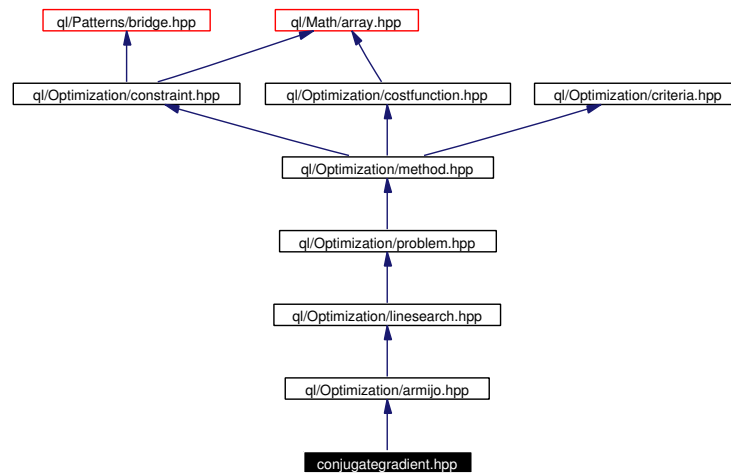
10.168 ql/Optimization/conjugategradient.hpp File Reference

10.168.1 Detailed Description

Conjugate gradient optimization method.

```
#include <ql/Optimization/armijo.hpp>
```

Include dependency graph for conjugategradient.hpp:



Namespaces

- namespace **QuantLib**

10.169 ql/Optimization/constraint.hpp File Reference

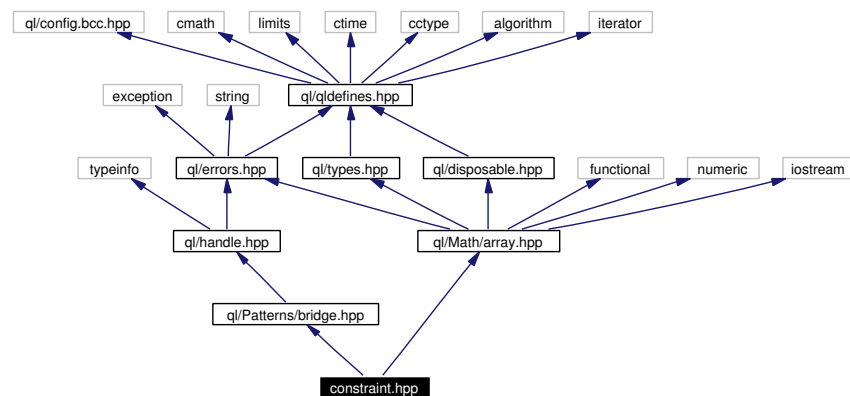
10.169.1 Detailed Description

Abstract constraint class.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Patterns/bridge.hpp>
```

Include dependency graph for constraint.hpp:



Namespaces

- namespace **QuantLib**

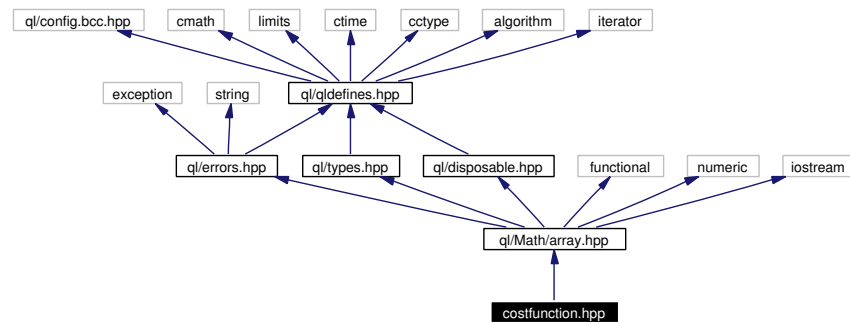
10.170 ql/Optimization/costfunction.hpp File Reference

10.170.1 Detailed Description

Optimization cost function class.

```
#include <ql/Math/array.hpp>
```

Include dependency graph for costfunction.hpp:



Namespaces

- namespace **QuantLib**

10.171 ql/Optimization/criteria.hpp File Reference

10.171.1 Detailed Description

Optimization criteria class.

Namespaces

- namespace **QuantLib**

10.172 ql/Optimization/leastsquare.hpp File Reference

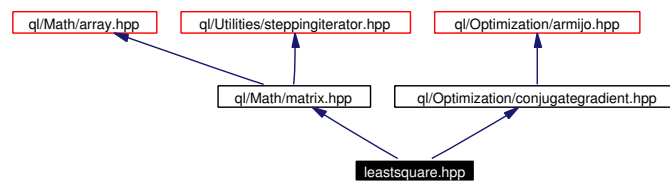
10.172.1 Detailed Description

Least square cost function.

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Optimization/conjugategradient.hpp>
```

Include dependency graph for leastsquare.hpp:



Namespaces

- namespace **QuantLib**

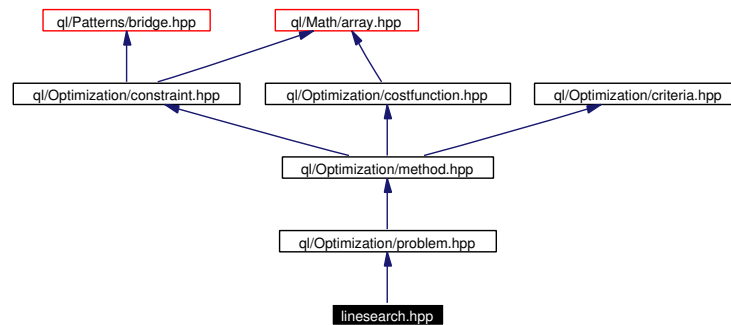
10.173 ql/Optimization/linesearch.hpp File Reference

10.173.1 Detailed Description

Line search abstract class.

```
#include <ql/Optimization/problem.hpp>
```

Include dependency graph for linesearch.hpp:



Namespaces

- namespace **QuantLib**

10.174 ql/Optimization/method.hpp File Reference

10.174.1 Detailed Description

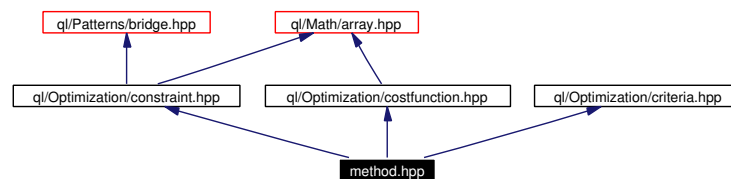
Abstract optimization method class.

```
#include <ql/Optimization/constraint.hpp>
```

```
#include <ql/Optimization/costfunction.hpp>
```

```
#include <ql/Optimization/criteria.hpp>
```

Include dependency graph for method.hpp:



Namespaces

- namespace **QuantLib**

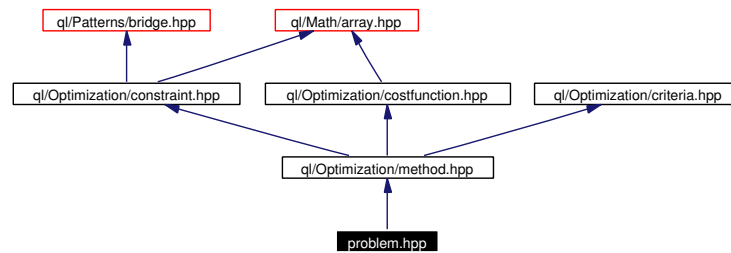
10.175 ql/Optimization/problem.hpp File Reference

10.175.1 Detailed Description

Abstract optimization class.

```
#include <ql/Optimization/method.hpp>
```

Include dependency graph for problem.hpp:



Namespaces

- namespace **QuantLib**

10.176 ql/Optimization/simplex.hpp File Reference

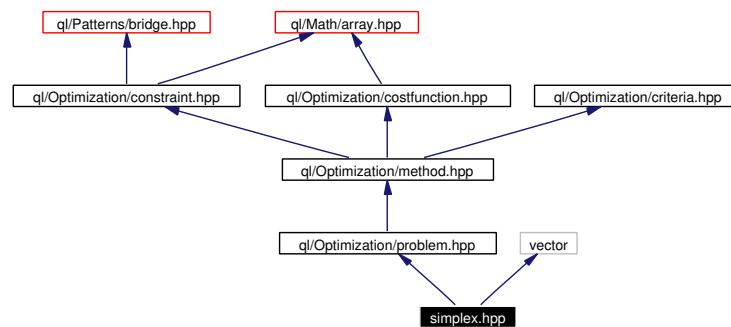
10.176.1 Detailed Description

Simplex optimization method.

```
#include <ql/Optimization/problem.hpp>
```

```
#include <vector>
```

Include dependency graph for simplex.hpp:



Namespaces

- namespace **QuantLib**

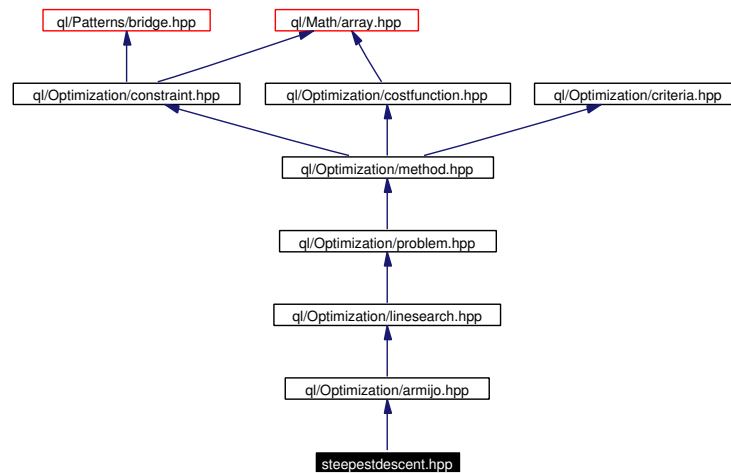
10.177 ql/Optimization/steepestdescent.hpp File Reference

10.177.1 Detailed Description

Steepest descent optimization method.

```
#include <ql/Optimization/armijo.hpp>
```

Include dependency graph for steepestdescent.hpp:



Namespaces

- namespace **QuantLib**

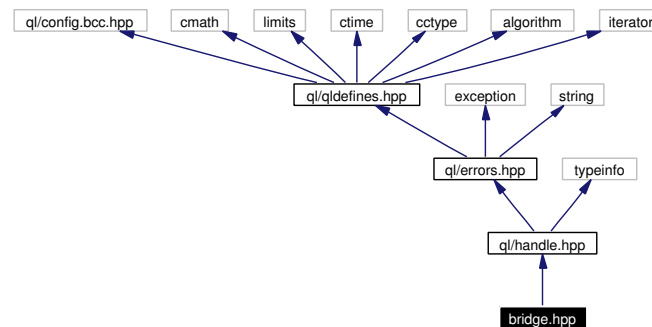
10.179 ql/Patterns/bridge.hpp File Reference

10.179.1 Detailed Description

bridge pattern (a.k.a. handle-body idiom)

```
#include <ql/handle.hpp>
```

Include dependency graph for bridge.hpp:



Namespaces

- namespace **QuantLib**

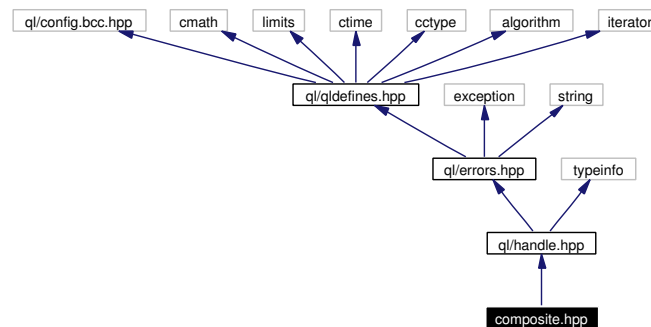
10.180 ql/Patterns/composite.hpp File Reference

10.180.1 Detailed Description

composite pattern

```
#include <ql/handle.hpp>
```

Include dependency graph for composite.hpp:



Namespaces

- namespace **QuantLib**

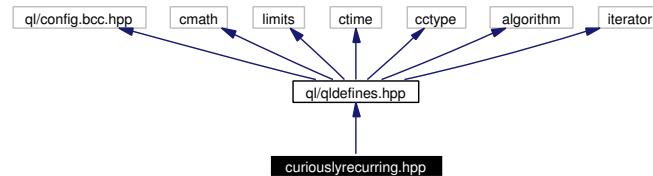
10.181 ql/Patterns/curiouslyrecurring.hpp File Reference

10.181.1 Detailed Description

Curiously recurring template pattern.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for curiouslyrecurring.hpp:



Namespaces

- namespace **QuantLib**

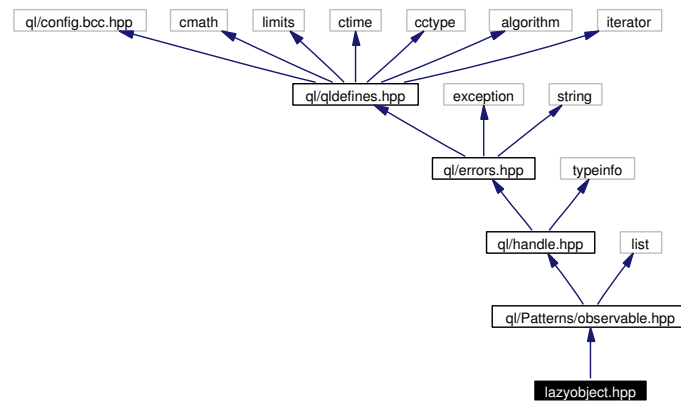
10.182 ql/Patterns/lazyobject.hpp File Reference

10.182.1 Detailed Description

framework for calculation on demand and result caching

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for lazyobject.hpp:



Namespaces

- namespace **QuantLib**

10.183 ql/Patterns/observable.hpp File Reference

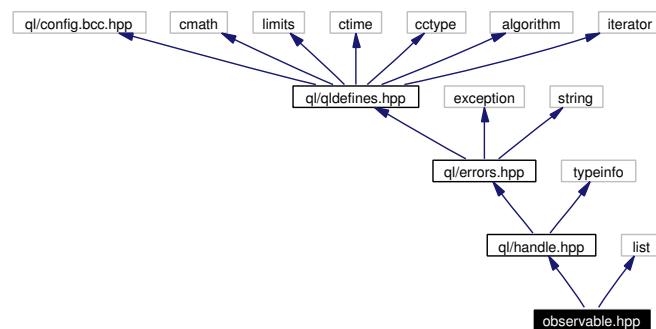
10.183.1 Detailed Description

observer/observable pattern

```
#include <ql/handle.hpp>
```

```
#include <list>
```

Include dependency graph for observable.hpp:



Namespaces

- namespace **QuantLib**

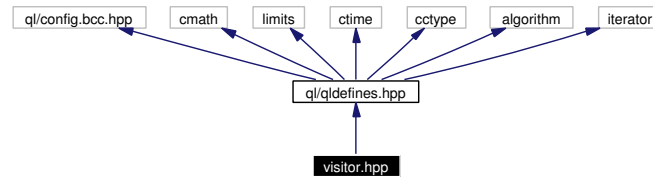
10.184 ql/Patterns/visitor.hpp File Reference

10.184.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

```
#include <ql/qldefines.hpp>
```

Include dependency graph for visitor.hpp:



Namespaces

- namespace **QuantLib**

10.185 ql/payoff.hpp File Reference

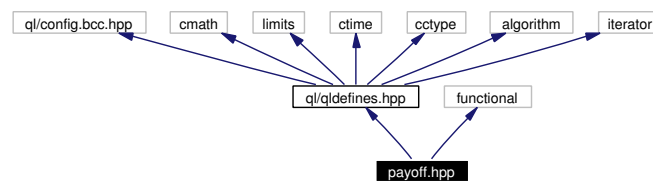
10.185.1 Detailed Description

Option payoff classes.

```
#include <ql/qldefines.hpp>
```

```
#include <functional>
```

Include dependency graph for payoff.hpp:



Namespaces

- namespace **QuantLib**

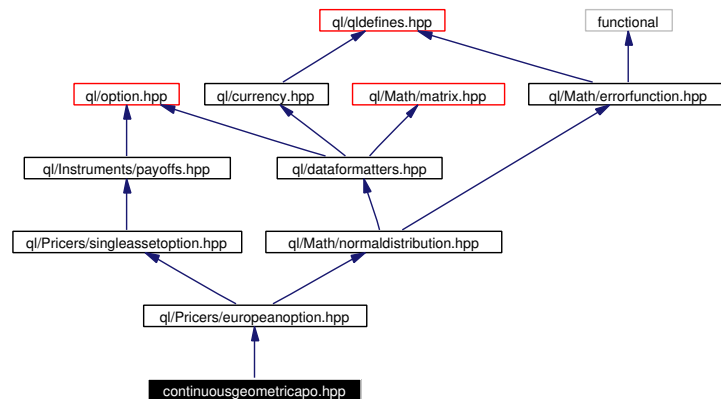
10.186 ql/Pricers/continuousgeometricapo.hpp File Reference

10.186.1 Detailed Description

Continuous Geometric Average Price Option (European exercise).

```
#include <ql/Pricers/europeanoption.hpp>
```

Include dependency graph for continuousgeometricapo.hpp:



Namespaces

- namespace **QuantLib**

10.187 ql/Pricers/discretegeometricapo.hpp File Reference

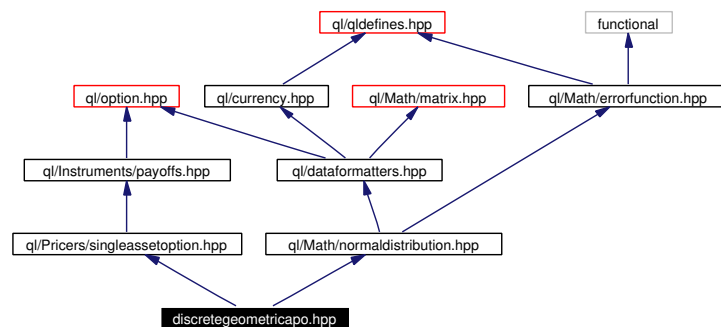
10.187.1 Detailed Description

Discrete Geometric Average Price Option.

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for discretegeometricapo.hpp:



Namespaces

- namespace **QuantLib**

10.188 ql/Pricers/discretegeometricaso.hpp File Reference

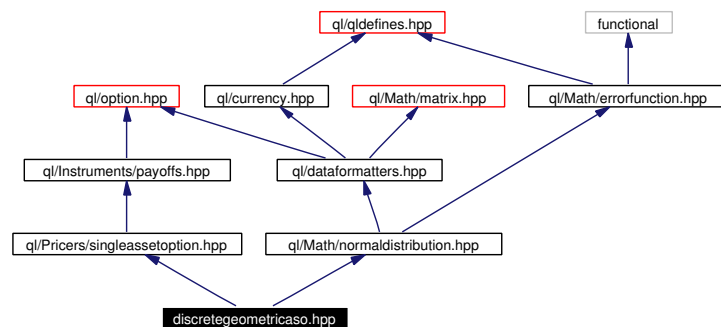
10.188.1 Detailed Description

Discrete Geometric Average Strike Option.

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for discretegeometricaso.hpp:



Namespaces

- namespace **QuantLib**

10.189 ql/Pricers/europeanoption.hpp File Reference

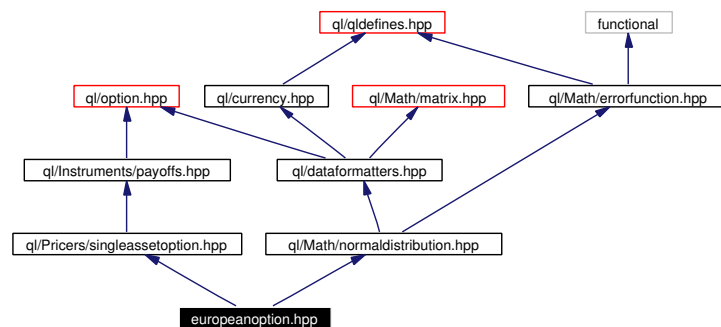
10.189.1 Detailed Description

european option

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for europeanoption.hpp:



Namespaces

- namespace **QuantLib**

10.190 ql/Pricers/fdamericanoption.hpp File Reference

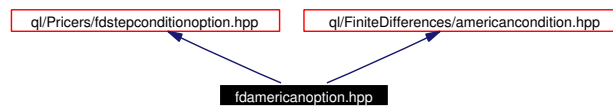
10.190.1 Detailed Description

american option

```
#include <ql/Pricers/fdstepconditionoption.hpp>
```

```
#include <ql/FiniteDifferences/americancondition.hpp>
```

Include dependency graph for fdamericanoption.hpp:



Namespaces

- namespace **QuantLib**

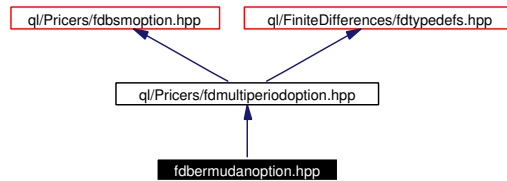
10.191 ql/Pricers/fdbermudanoption.hpp File Reference

10.191.1 Detailed Description

finite-difference evaluation of Bermudan option

```
#include <ql/Pricers/fdmultipleriodoption.hpp>
```

Include dependency graph for fdbermudanoption.hpp:



Namespaces

- namespace **QuantLib**

10.192 ql/Pricers/fdbsmoption.hpp File Reference

10.192.1 Detailed Description

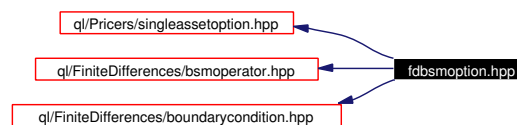
common code for numerical option evaluation

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/FiniteDifferences/bsmoperator.hpp>
```

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Include dependency graph for fdbsmoption.hpp:



Namespaces

- namespace **QuantLib**

Defines

- `#define QL_NUM_OPT_MIN_GRID_POINTS 10`
This is a safety check to be sure we have enough grid points.
- `#define QL_NUM_OPT_GRID_POINTS_PER_YEAR 2`
This is a safety check to be sure we have enough grid points.

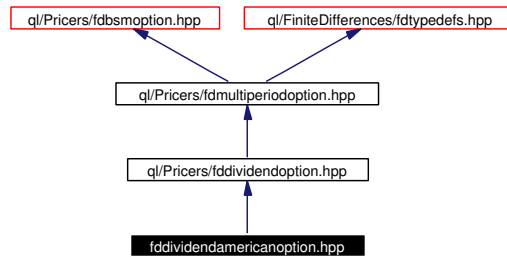
10.193 ql/Pricers/fddividendamericanoption.hpp File Reference

10.193.1 Detailed Description

american option with discrete deterministic dividends

```
#include <ql/Pricers/fddividendoption.hpp>
```

Include dependency graph for fddividendamericanoption.hpp:



Namespaces

- namespace **QuantLib**

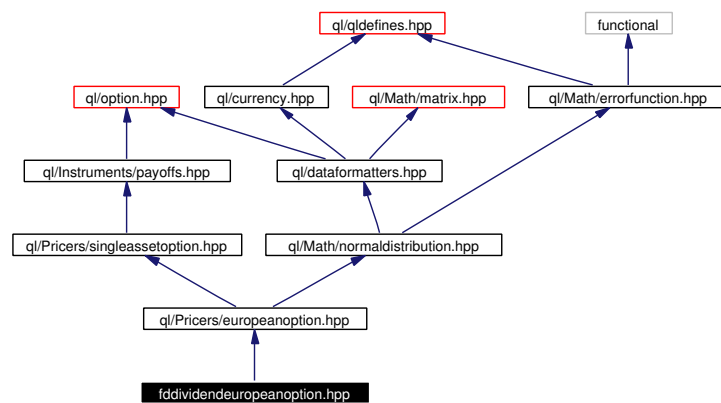
10.194 ql/Pricers/fddividendeuropeanoption.hpp File Reference

10.194.1 Detailed Description

european option with discrete deterministic dividends

```
#include <ql/Pricers/europeanoption.hpp>
```

Include dependency graph for fddividendeuropeanoption.hpp:



Namespaces

- namespace **QuantLib**

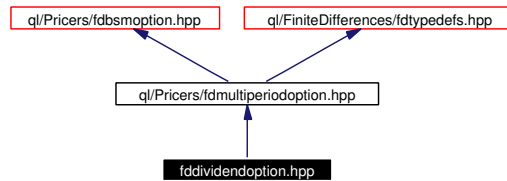
10.195 ql/Pricers/fddividendoption.hpp File Reference

10.195.1 Detailed Description

base class for option with dividends

```
#include <ql/Pricers/fdmultipleriodoption.hpp>
```

Include dependency graph for fddividendoption.hpp:



Namespaces

- namespace **QuantLib**

10.196 ql/Pricers/fddividendshoutoption.hpp File Reference

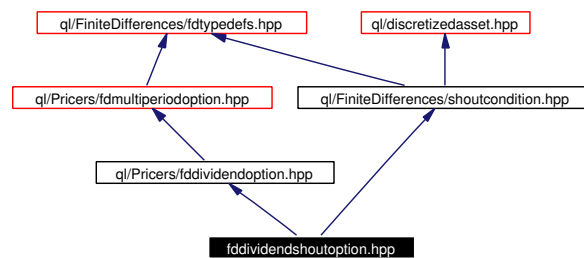
10.196.1 Detailed Description

base class for shout option with dividends

```
#include <ql/Pricers/fddividendoption.hpp>
```

```
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

Include dependency graph for fddividendshoutoption.hpp:



Namespaces

- namespace **QuantLib**

10.197 ql/Pricers/fdeuropean.hpp File Reference

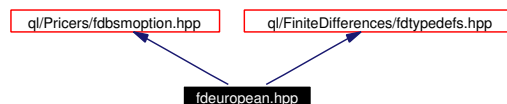
10.197.1 Detailed Description

Example of European option calculated using finite differences.

```
#include <ql/Pricers/fdbsmoption.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

Include dependency graph for fdeuropean.hpp:



Namespaces

- namespace **QuantLib**

10.198 ql/Pricers/fdmultiperiodoption.hpp File Reference

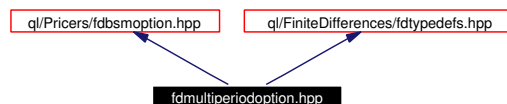
10.198.1 Detailed Description

base class for option with events happening at different periods

```
#include <ql/Pricers/fdbsmoption.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

Include dependency graph for fdmultiperiodoption.hpp:



Namespaces

- namespace **QuantLib**

10.199 ql/Pricers/fdshoutoption.hpp File Reference

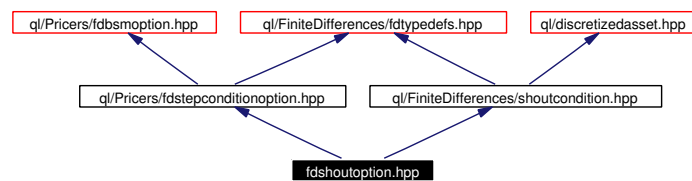
10.199.1 Detailed Description

shout option

```
#include <ql/Pricers/fdstepconditionoption.hpp>
```

```
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

Include dependency graph for fdshoutoption.hpp:



Namespaces

- namespace **QuantLib**

10.200 ql/Pricers/fdstepconditionoption.hpp File Reference

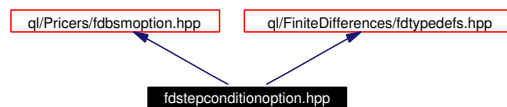
10.200.1 Detailed Description

Option requiring additional code to be executed at each time step.

```
#include <ql/Pricers/fdbsmoption.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

Include dependency graph for fdstepconditionoption.hpp:



Namespaces

- namespace **QuantLib**

10.201 ql/Pricers/mcbasket.hpp File Reference

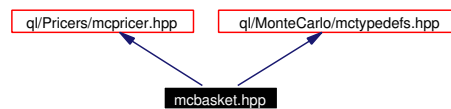
10.201.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mcbasket.hpp:



Namespaces

- namespace **QuantLib**

10.202 ql/Pricers/mccliquestoption.hpp File Reference

10.202.1 Detailed Description

Cliquet option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mccliquestoption.hpp:



Namespaces

- namespace **QuantLib**

10.203 ql/Pricers/mcdiscretearithmeticapo.hpp File Reference

10.203.1 Detailed Description

Discrete Arithmetic Average Price Option.

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mcdiscretearithmeticapo.hpp:



Namespaces

- namespace **QuantLib**

10.204 ql/Pricers/mcdiscretearithmeticaso.hpp File Reference

10.204.1 Detailed Description

Discrete Arithmetic Average Strike Option.

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mcdiscretearithmeticaso.hpp:



Namespaces

- namespace **QuantLib**

10.205 ql/Pricers/mceverest.hpp File Reference

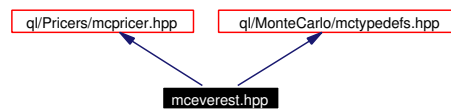
10.205.1 Detailed Description

Everest-type option pricer

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mceverest.hpp:



Namespaces

- namespace **QuantLib**

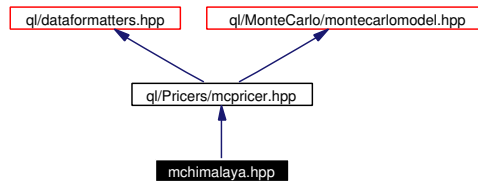
10.206 ql/Pricers/mchimalaya.hpp File Reference

10.206.1 Detailed Description

Himalayan-type option pricer.

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mchimalaya.hpp:



Namespaces

- namespace **QuantLib**

10.207 ql/Pricers/mcmaxbasket.hpp File Reference

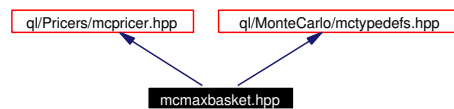
10.207.1 Detailed Description

Max Basket Monte Carlo pricer.

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mcmaxbasket.hpp:



Namespaces

- namespace **QuantLib**

10.208 ql/Pricers/mcpagoda.hpp File Reference

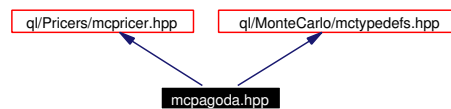
10.208.1 Detailed Description

Roofed multi asset Asian option.

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mcpagoda.hpp:



Namespaces

- namespace **QuantLib**

10.209 ql/Pricers/mcperformanceoption.hpp File Reference

10.209.1 Detailed Description

Performance option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

```
#include <ql/MonteCarlo/mctypedefs.hpp>
```

Include dependency graph for mcperformanceoption.hpp:



Namespaces

- namespace **QuantLib**

10.210 ql/Pricers/mcpricer.hpp File Reference

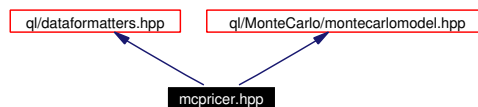
10.210.1 Detailed Description

base class for Monte Carlo pricers

```
#include <ql/dataformatters.hpp>
```

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

Include dependency graph for mcpricer.hpp:



Namespaces

- namespace **QuantLib**

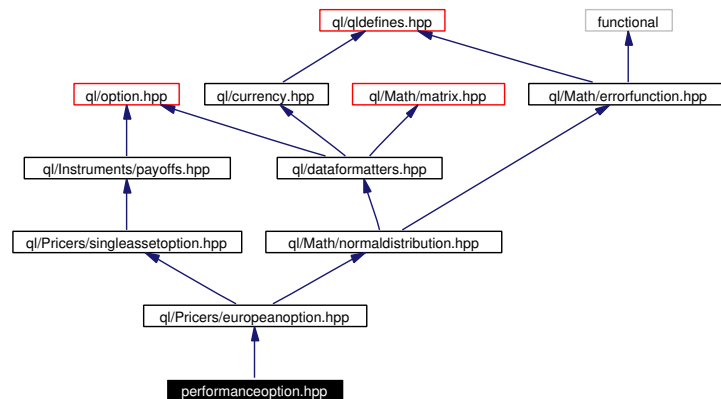
10.211 ql/Pricers/performanceoption.hpp File Reference

10.211.1 Detailed Description

Performance option.

```
#include <ql/Pricers/europeanoption.hpp>
```

Include dependency graph for performanceoption.hpp:



Namespaces

- namespace **QuantLib**

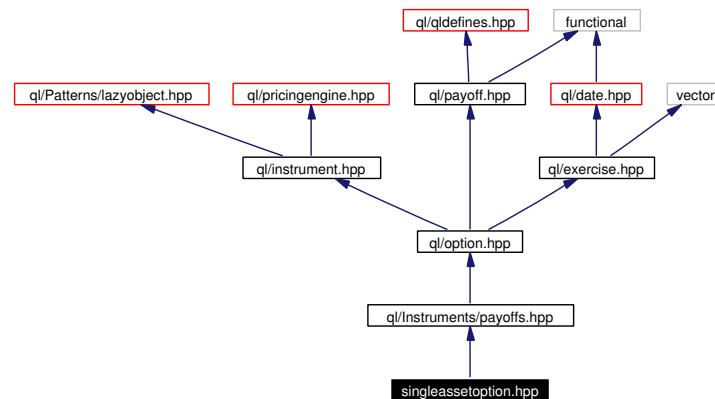
10.212 ql/Pricers/singleassetoption.hpp File Reference

10.212.1 Detailed Description

common code for option evaluation

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for singleassetoption.hpp:



Namespaces

- namespace **QuantLib**

10.213 ql/pricingengine.hpp File Reference

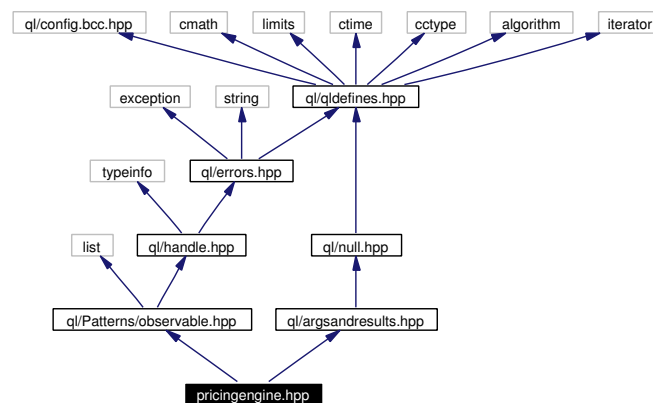
10.213.1 Detailed Description

Base class for pricing engines.

```
#include <ql/argsandresults.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for pricingengine.hpp:



Namespaces

- namespace **QuantLib**

10.214 ql/PricingEngines/americanpayoffatexpiry.hpp File Reference

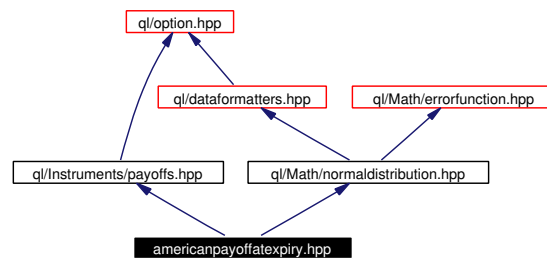
10.214.1 Detailed Description

Analytical formulae for american exercise with payoff at expiry.

```
#include <ql/Instruments/payoffs.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for americanpayoffatexpiry.hpp:



Namespaces

- namespace **QuantLib**

10.215 ql/PricingEngines/americanpayoffathit.hpp File Reference

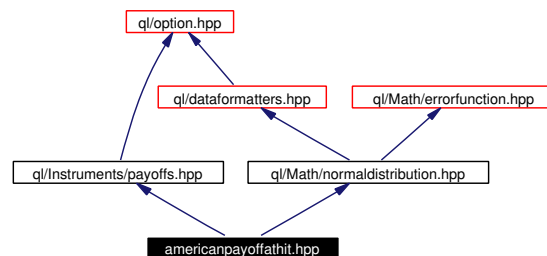
10.215.1 Detailed Description

Analytical formulae for american exercise with payoff at hit.

```
#include <ql/Instruments/payoffs.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for americanpayoffathit.hpp:



Namespaces

- namespace **QuantLib**

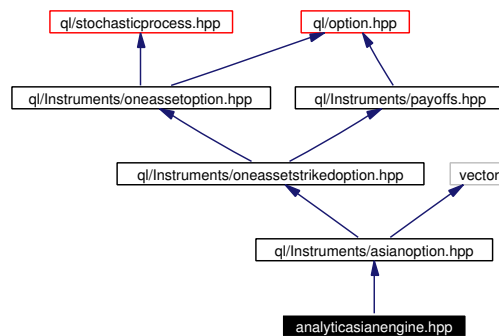
10.216 ql/PricingEngines/Asian/analyticasianengine.hpp File Reference

10.216.1 Detailed Description

Analytic Asian option engine.

```
#include <ql/Instruments/asianoption.hpp>
```

Include dependency graph for analyticasianengine.hpp:



Namespaces

- namespace **QuantLib**

10.217 ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference

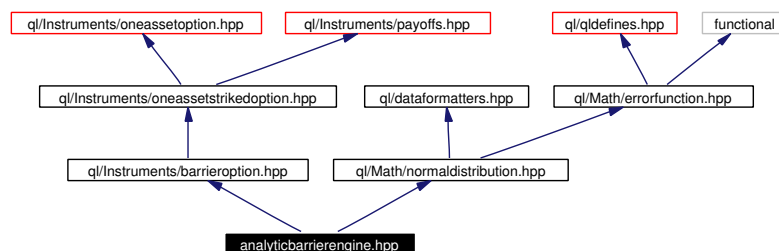
10.217.1 Detailed Description

Analytic barrier option engines.

```
#include <ql/Instruments/barrieroption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for analyticbarrierengine.hpp:



Namespaces

- namespace **QuantLib**

10.218 ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference

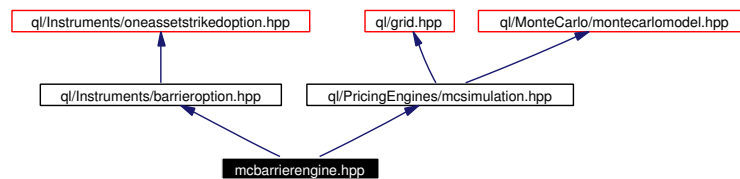
10.218.1 Detailed Description

Monte Carlo barrier option engines.

```
#include <ql/Instruments/barrieroption.hpp>
```

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

Include dependency graph for mcbarrierengine.hpp:



Namespaces

- namespace **QuantLib**

10.219 ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference

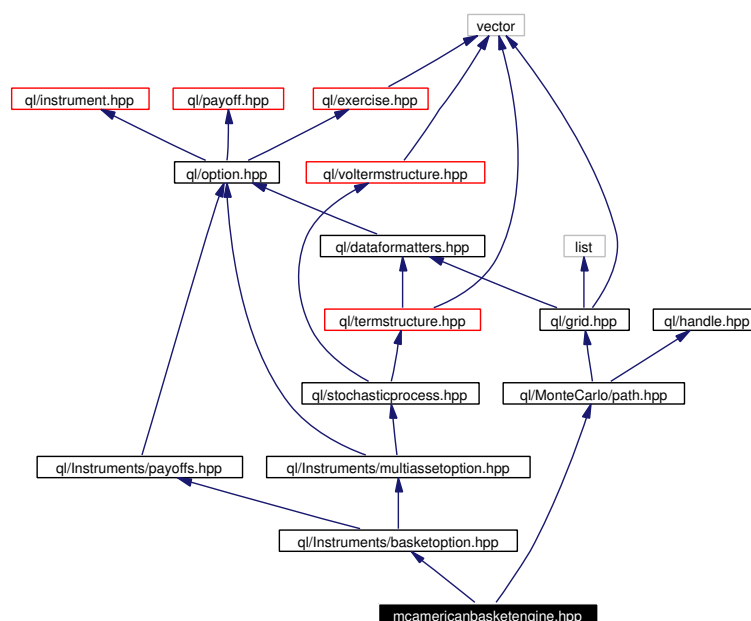
10.219.1 Detailed Description

Least-square Monte Carlo engines.

```
#include <ql/Instruments/basketoption.hpp>
```

```
#include <ql/MonteCarlo/path.hpp>
```

Include dependency graph for mcamericanbasketengine.hpp:



Namespaces

- namespace **QuantLib**

10.220 ql/PricingEngines/Basket/mcbasketengine.hpp File Reference

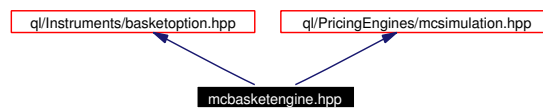
10.220.1 Detailed Description

European Basket MC Engine.

```
#include <ql/Instruments/basketoption.hpp>
```

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

Include dependency graph for mcbasketengine.hpp:



Namespaces

- namespace **QuantLib**

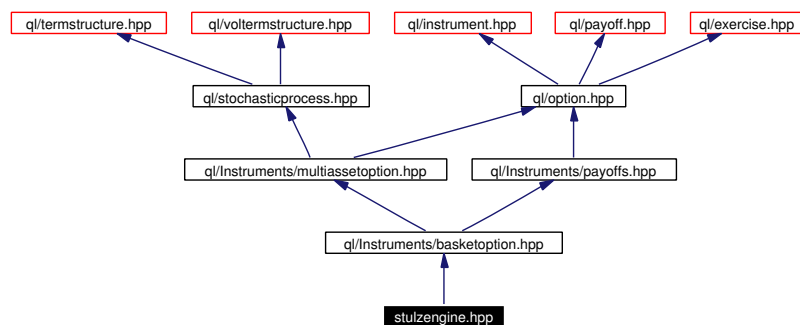
10.221 ql/PricingEngines/Basket/stulzengine.hpp File Reference

10.221.1 Detailed Description

2D European Basket formulae, due to Stulz (1982)

```
#include <ql/Instruments/basketoption.hpp>
```

Include dependency graph for stulzengine.hpp:



Namespaces

- namespace **QuantLib**

10.222 ql/PricingEngines/blackformula.hpp File Reference

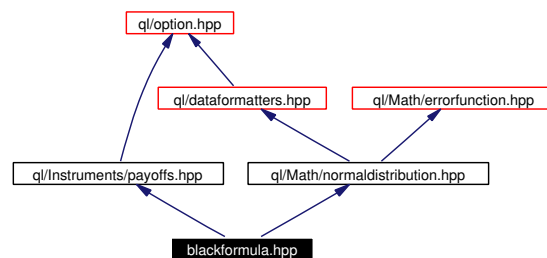
10.222.1 Detailed Description

Black formula.

```
#include <ql/Instruments/payoffs.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for blackformula.hpp:



Namespaces

- namespace **QuantLib**

10.223 ql/PricingEngines/blackmodel.hpp File Reference

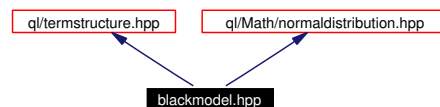
10.223.1 Detailed Description

Abstract class for Black-type models (market models).

```
#include <ql/termstructure.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for blackmodel.hpp:



Namespaces

- namespace **QuantLib**

10.224 ql/PricingEngines/CapFloor/analyticalcapfloor.hpp File Reference

10.224.1 Detailed Description

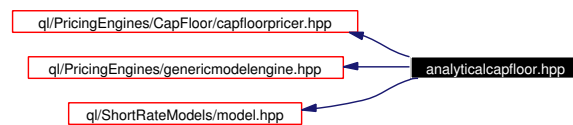
Analytical pricer for caps/floors.

```
#include <ql/PricingEngines/CapFloor/capfloorpricer.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

```
#include <ql/ShortRateModels/model.hpp>
```

Include dependency graph for analyticalcapfloor.hpp:



Namespaces

- namespace **QuantLib**

10.225 ql/PricingEngines/CapFloor/blackcapfloor.hpp File Reference

10.225.1 Detailed Description

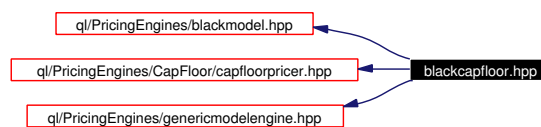
CapFloor calculated using the Black formula.

```
#include <ql/PricingEngines/blackmodel.hpp>
```

```
#include <ql/PricingEngines/CapFloor/capfloorpricer.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for blackcapfloor.hpp:



Namespaces

- namespace **QuantLib**

10.226 ql/PricingEngines/CapFloor/capfloorpricer.hpp File Reference

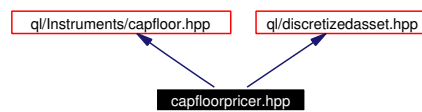
10.226.1 Detailed Description

cap and floor pricer class

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for capfloorpricer.hpp:



Namespaces

- namespace **QuantLib**

10.227 ql/PricingEngines/CapFloor/treecapfloor.hpp File Reference

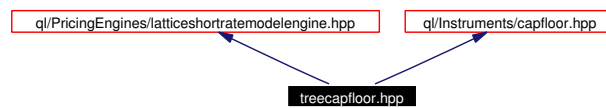
10.227.1 Detailed Description

Cap/Floor calculated using a tree.

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

```
#include <ql/Instruments/capfloor.hpp>
```

Include dependency graph for treecapfloor.hpp:



Namespaces

- namespace **QuantLib**

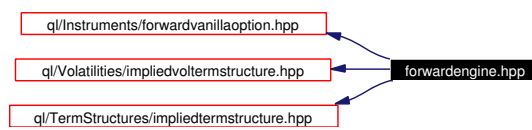
10.228 ql/PricingEngines/Forward/forwardengine.hpp File Reference

10.228.1 Detailed Description

Forward (strike-resetting) option engine.

```
#include <ql/Instruments/forwardvanillaoption.hpp>
#include <ql/Volatilities/impliedvoltermstructure.hpp>
#include <ql/TermStructures/impliedtermstructure.hpp>
```

Include dependency graph for forwardengine.hpp:



Namespaces

- namespace **QuantLib**

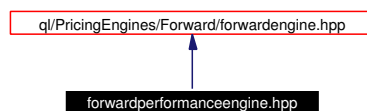
10.229 ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference

10.229.1 Detailed Description

Forward (strike-resetting) performance option engines.

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

Include dependency graph for forwardperformanceengine.hpp:



Namespaces

- namespace **QuantLib**

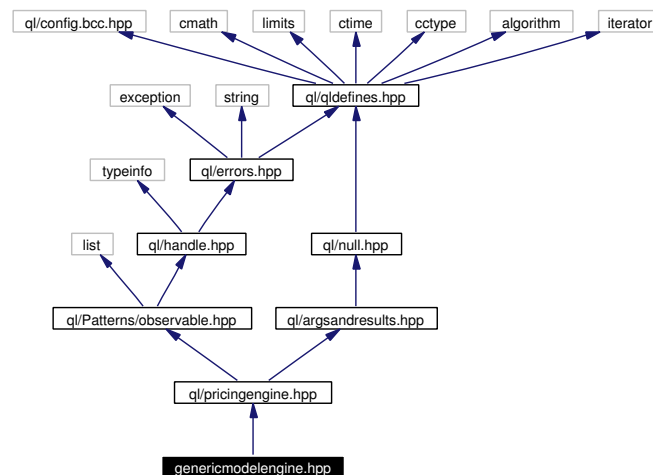
10.230 ql/PricingEngines/genericmodelengine.hpp File Reference

10.230.1 Detailed Description

Generic option engine based on a model.

```
#include <ql/pricingengine.hpp>
```

Include dependency graph for genericmodelengine.hpp:



Namespaces

- namespace **QuantLib**

10.231 ql/PricingEngines/latticeshortratemodelengine.hpp File Reference

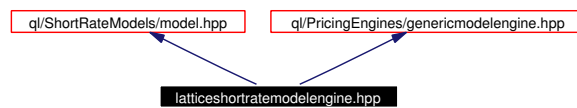
10.231.1 Detailed Description

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**

10.232 ql/PricingEngines/mcsimulation.hpp File Reference

10.232.1 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**

10.233 ql/PricingEngines/Quanto/quantoengine.hpp File Reference

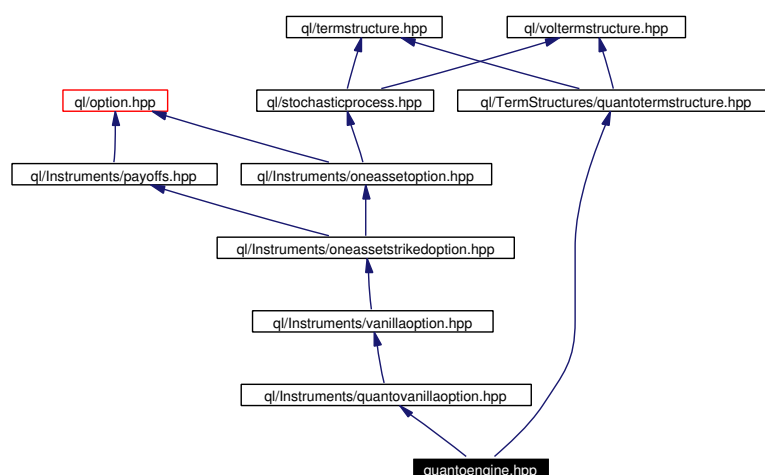
10.233.1 Detailed Description

Quanto option engine.

```
#include <ql/Instruments/quantovanillaoption.hpp>
```

```
#include <ql/TermStructures/quantotermstructure.hpp>
```

Include dependency graph for quantoengine.hpp:



Namespaces

- namespace **QuantLib**

10.234 ql/PricingEngines/Swaption/blackswaption.hpp File Reference

10.234.1 Detailed Description

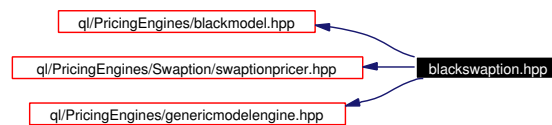
Swaption calculated using the Black formula.

```
#include <ql/PricingEngines/blackmodel.hpp>
```

```
#include <ql/PricingEngines/Swaption/swaptionpricer.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for blackswaption.hpp:



Namespaces

- namespace **QuantLib**

10.235 ql/PricingEngines/Swaption/jamshidianswaption.hpp File Reference

10.235.1 Detailed Description

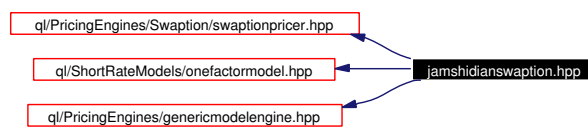
Swaption pricer using Jamshidian's decomposition.

```
#include <ql/PricingEngines/Swaption/swaptionpricer.hpp>
```

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for jamshidianswaption.hpp:



Namespaces

- namespace **QuantLib**

10.236 ql/PricingEngines/Swaption/swaptionpricer.hpp File Reference

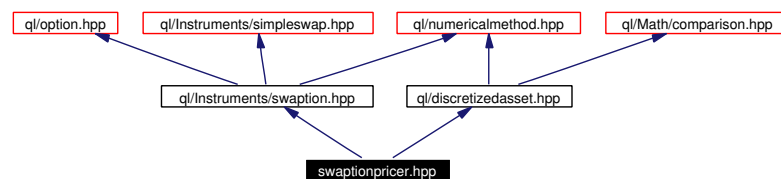
10.236.1 Detailed Description

Swaption pricer class.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for swaptionpricer.hpp:



Namespaces

- namespace **QuantLib**

10.237 ql/PricingEngines/Swaption/treeswaption.hpp File Reference

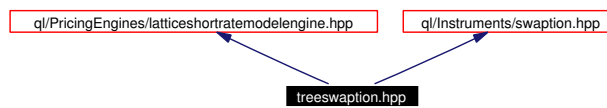
10.237.1 Detailed Description

Swaption computed using a lattice.

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

```
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for treeswaption.hpp:



Namespaces

- namespace **QuantLib**

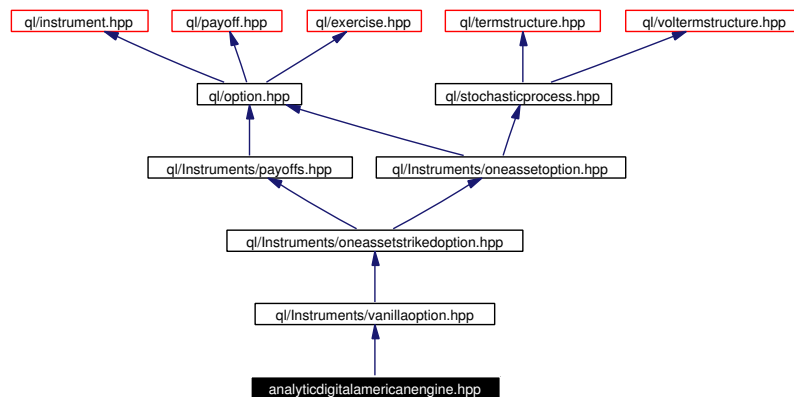
10.238 ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference

10.238.1 Detailed Description

analytic digital American option engine

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for analyticdigitalamericanengine.hpp:



Namespaces

- namespace **QuantLib**

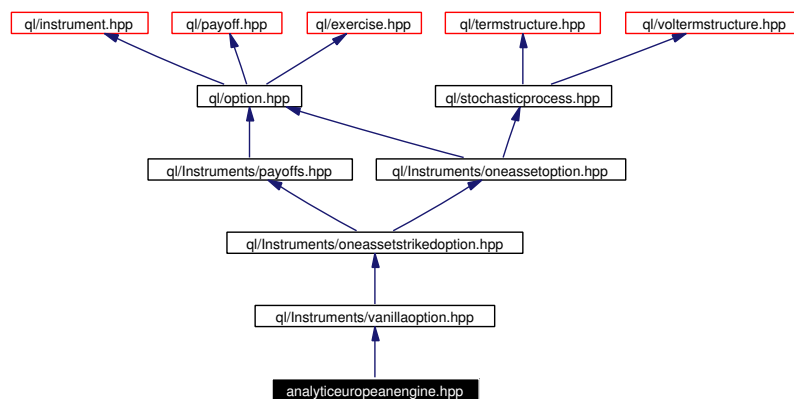
10.239 ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference

10.239.1 Detailed Description

Analytic European engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for `analyticeuropeanengine.hpp`:



Namespaces

- namespace **QuantLib**

10.241 ql/PricingEngines/Vanilla/binomialengine.hpp File Reference

10.241.1 Detailed Description

Binomial option engine.

```
#include <ql/Lattices/binomialtree.hpp>
```

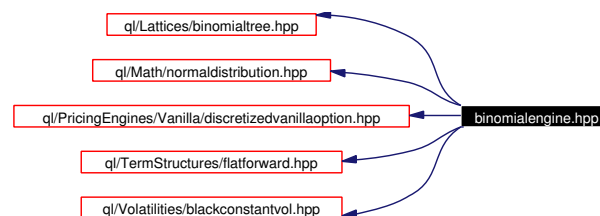
```
#include <ql/Math/normaldistribution.hpp>
```

```
#include <ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp>
```

```
#include <ql/TermStructures/flatforward.hpp>
```

```
#include <ql/Volatilities/blackconstantvol.hpp>
```

Include dependency graph for binomialengine.hpp:



Namespaces

- namespace **QuantLib**

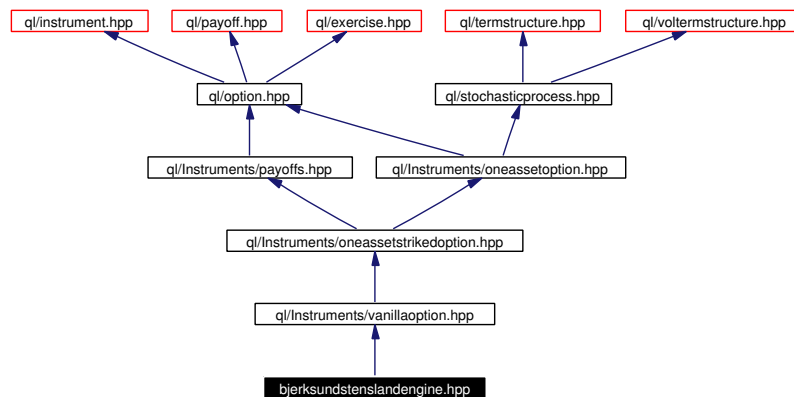
10.242 ql/PricingEngines/Vanilla/bjersundstenslandengine.hpp File Reference

10.242.1 Detailed Description

Bjersund and Stensland approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for bjersundstenslandengine.hpp:



Namespaces

- namespace **QuantLib**

10.243 ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference

10.243.1 Detailed Description

discretized vanilla option

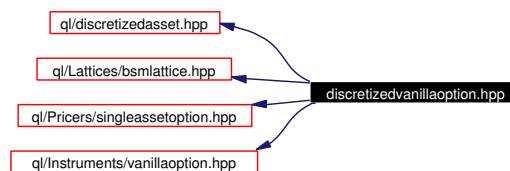
```
#include <ql/discretizedasset.hpp>
```

```
#include <ql/Lattices/bsmlattice.hpp>
```

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for discretizedvanillaoption.hpp:



Namespaces

- namespace **QuantLib**

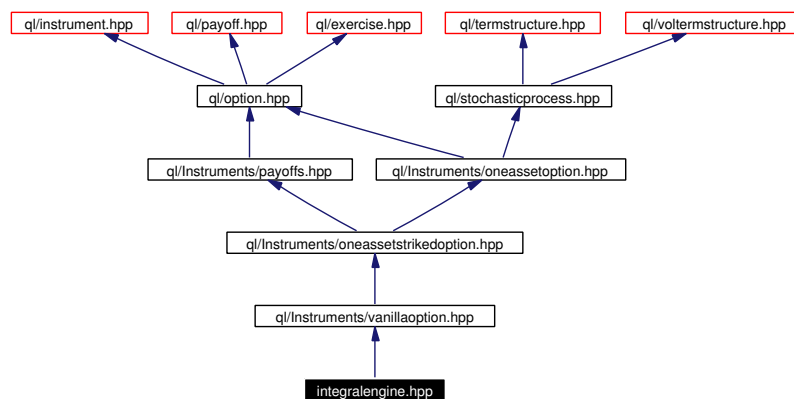
10.244 ql/PricingEngines/Vanilla/integralengine.hpp File Reference

10.244.1 Detailed Description

Integral option engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for integralengine.hpp:



Namespaces

- namespace **QuantLib**

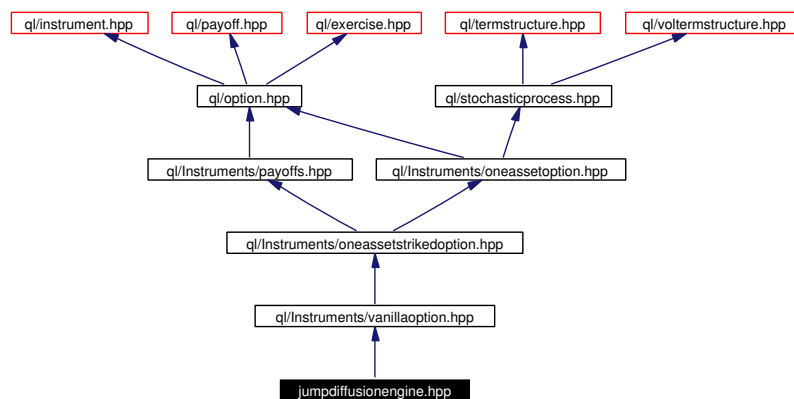
10.245 ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference

10.245.1 Detailed Description

Jump diffusion (Merton 1976) engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for jumpdiffusionengine.hpp:



Namespaces

- namespace **QuantLib**

10.246 ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference

10.246.1 Detailed Description

digital option Monte Carlo engine

```
#include <ql/exercise.hpp>
```

```
#include <ql/handle.hpp>
```

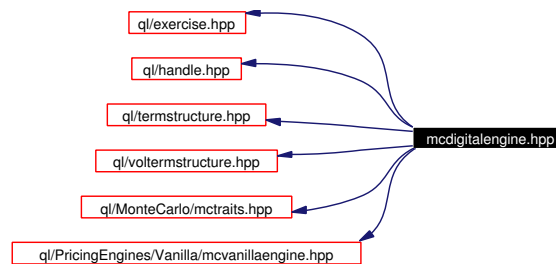
```
#include <ql/termstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/MonteCarlo/mctrails.hpp>
```

```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

Include dependency graph for mcdigitalengine.hpp:



Namespaces

- namespace **QuantLib**

10.247 ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference

10.247.1 Detailed Description

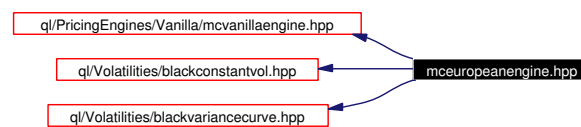
Monte Carlo European option engine.

```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

```
#include <ql/Volatilities/blackconstantvol.hpp>
```

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for mceuropeanengine.hpp:



Namespaces

- namespace **QuantLib**

10.248 ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference

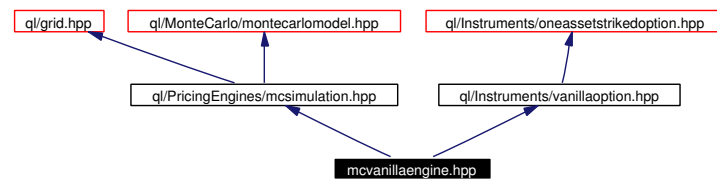
10.248.1 Detailed Description

Monte Carlo vanilla option engine.

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for mcvanillaengine.hpp:



Namespaces

- namespace **QuantLib**

10.249 ql/qldefines.hpp File Reference

10.249.1 Detailed Description

Global definitions and compiler switches.

```
#include <ql/config.bcc.hpp>
```

```
#include <cmath>
```

```
#include <limits>
```

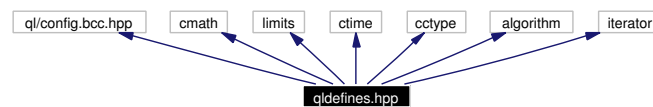
```
#include <ctime>
```

```
#include <cctype>
```

```
#include <algorithm>
```

```
#include <iterator>
```

Include dependency graph for qldefines.hpp:



Namespaces

- namespace **QuantLib**

Defines

- **#define QL_HEX_VERSION** 0x000306f0
version hexadecimal number
- **#define QL_VERSION** "0.3.6"
version string
- **#define QL_DUMMY_RETURN(x)**
specific per-compiler definitions Is a dummy return statement required?
- **#define QL_IO_INIT**
I/O initialization.
- **#define QL_MIN_INT** ((std::numeric_limits<int>::min)())
- **#define QL_MAX_INT** ((std::numeric_limits<int>::max)())
- **#define QL_MIN_DOUBLE** -((std::numeric_limits<double>::max)())
- **#define QL_MAX_DOUBLE** ((std::numeric_limits<double>::max)())
- **#define QL_EPSILON** ((std::numeric_limits<double>::epsilon)())
- **#define QL_MIN_POSITIVE_DOUBLE** ((std::numeric_limits<double>::min)())
- **#define QL_DECLARE_TEMPLATE_SPECIALIZATIONS**
Blame Microsoft for this one...

- **#define QL_ALLOW_TEMPLATE_METHOD_CALLS 1**
Blame Microsoft for this one...
- **#define QL_TYPENAME typename**
Blame Microsoft for this one...
- **#define QL_TEMPLATE_METAPROGRAMMING_WORKS 1**
- **#define QL_SPECIALIZE_ITERATOR_TRAITS(T)**
- **#define QL_REVERSE_ITERATOR(iterator, type) std::reverse_iterator< iterator >**
Blame Microsoft for this one...
- **#define QL_FULL_ITERATOR_SUPPORT**

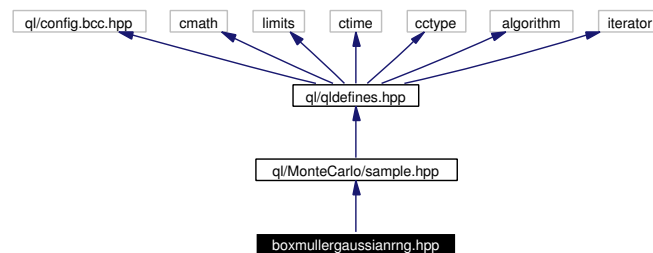
10.250 ql/RandomNumbers/boxmullergaussianrng.hpp File Reference

10.250.1 Detailed Description

Box-Muller Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for boxmullergaussianrng.hpp:



Namespaces

- namespace **QuantLib**

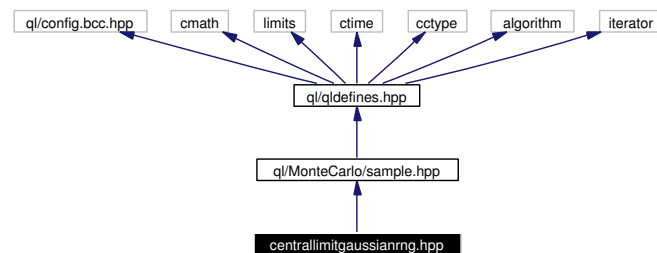
10.251 ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference

10.251.1 Detailed Description

Central limit Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for centrallimitgaussianrng.hpp:



Namespaces

- namespace **QuantLib**

10.252 ql/RandomNumbers/haltonrsg.hpp File Reference

10.252.1 Detailed Description

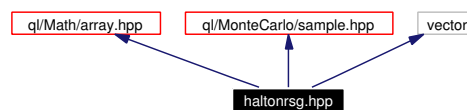
Halton low-discrepancy sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for haltonrsg.hpp:



Namespaces

- namespace **QuantLib**

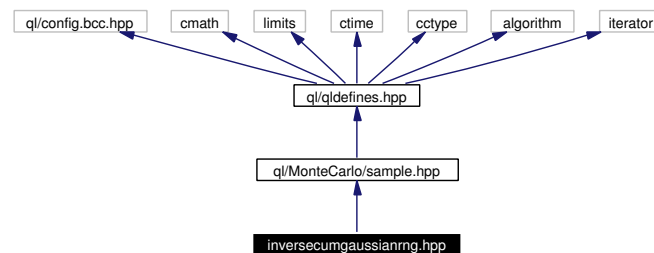
10.253 ql/RandomNumbers/inversecumgaussianrng.hpp File Reference

10.253.1 Detailed Description

Inverse cumulative Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for inversecumgaussianrng.hpp:



Namespaces

- namespace **QuantLib**

10.254 ql/RandomNumbers/inversecumgaussianrsg.hpp File Reference

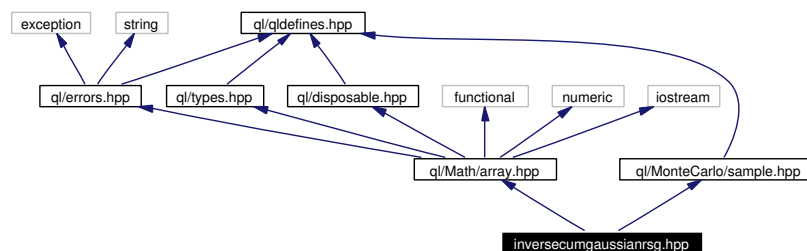
10.254.1 Detailed Description

Inverse cumulative Gaussian random sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for inversecumgaussianrsg.hpp:



Namespaces

- namespace **QuantLib**

10.255 ql/RandomNumbers/knuthuniformrng.hpp File Reference

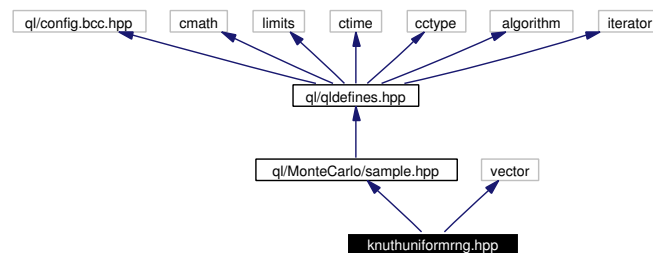
10.255.1 Detailed Description

Knuth uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for knuthuniformrng.hpp:



Namespaces

- namespace **QuantLib**

10.256 ql/RandomNumbers/lecuyeruniformrng.hpp File Reference

10.256.1 Detailed Description

L'Ecuyer uniform random number generator.

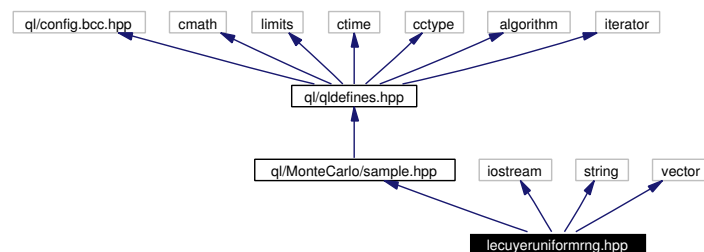
```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

Include dependency graph for lecuyeruniformrng.hpp:



Namespaces

- namespace **QuantLib**

10.257 ql/RandomNumbers/mt19937uniformrng.hpp File Reference

10.257.1 Detailed Description

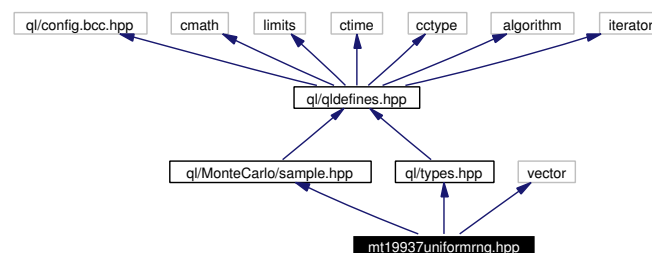
Mersenne Twister uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <vector>
```

Include dependency graph for mt19937uniformrng.hpp:



Namespaces

- namespace **QuantLib**

10.258 ql/RandomNumbers/randomarraygenerator.hpp File Reference

10.258.1 Detailed Description

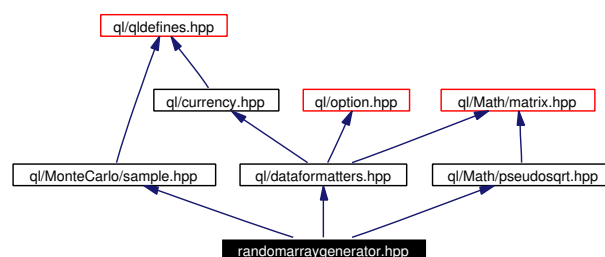
Generates random arrays from a random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <ql/Math/pseudosqrt.hpp>
```

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for randomarraygenerator.hpp:



Namespaces

- namespace **QuantLib**

10.259 ql/RandomNumbers/randomsequencegenerator.hpp File Reference

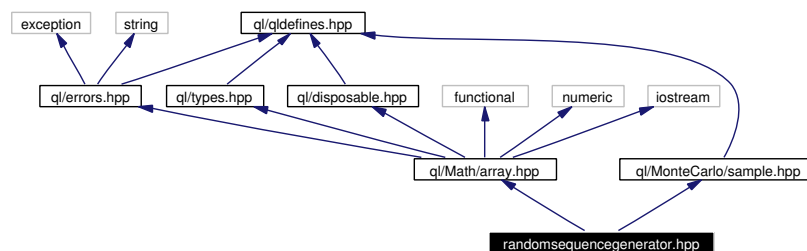
10.259.1 Detailed Description

Random sequence generator based on a pseudo-random number generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for randomsequencegenerator.hpp:



Namespaces

- namespace **QuantLib**

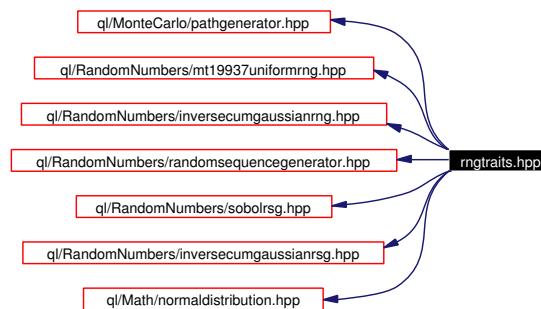
10.260 ql/RandomNumbers/rngtraits.hpp File Reference

10.260.1 Detailed Description

random-number generation policies

```
#include <ql/MonteCarlo/pathgenerator.hpp>
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
#include <ql/RandomNumbers/inversecumgaussianrng.hpp>
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
#include <ql/RandomNumbers/sobolrsg.hpp>
#include <ql/RandomNumbers/inversecumgaussianrsg.hpp>
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for rngtraits.hpp:



Namespaces

- namespace **QuantLib**

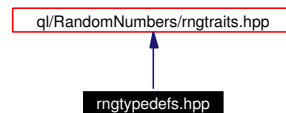
10.261 ql/RandomNumbers/rngtypedefs.hpp File Reference

10.261.1 Detailed Description

Default choices for template instantiations.

```
#include <ql/RandomNumbers/rngtraits.hpp>
```

Include dependency graph for rngtypedefs.hpp:



Namespaces

- namespace **QuantLib**

10.262 ql/RandomNumbers/sobolrsg.hpp File Reference

10.262.1 Detailed Description

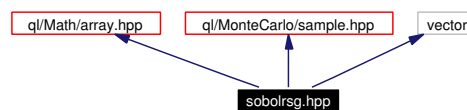
Sobol low-discrepancy sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for sobolrsg.hpp:



Namespaces

- namespace **QuantLib**

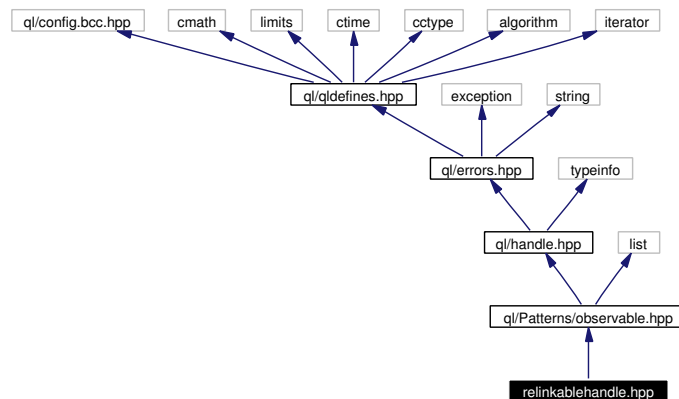
10.263 ql/relinkablehandle.hpp File Reference

10.263.1 Detailed Description

Globally accessible relinkable pointer.

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for relinkablehandle.hpp:



Namespaces

- namespace **QuantLib**

10.264 ql/scheduler.hpp File Reference

10.264.1 Detailed Description

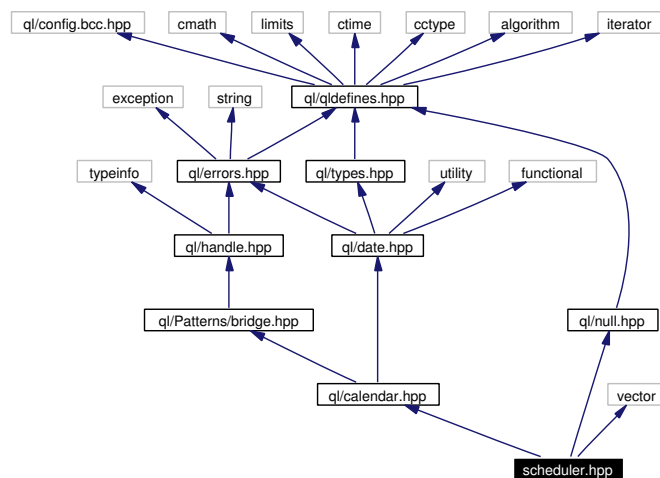
date scheduler

```
#include <ql/calendar.hpp>
```

```
#include <ql/null.hpp>
```

```
#include <vector>
```

Include dependency graph for scheduler.hpp:



Namespaces

- namespace **QuantLib**

10.265 ql/ShortRateModels/calibrationhelper.hpp File Reference

10.265.1 Detailed Description

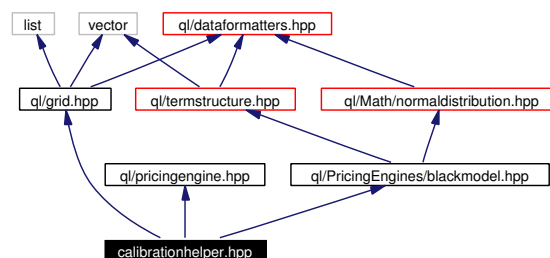
Calibration helper class.

```
#include <ql/grid.hpp>
```

```
#include <ql/pricingengine.hpp>
```

```
#include <ql/PricingEngines/blackmodel.hpp>
```

Include dependency graph for calibrationhelper.hpp:



Namespaces

- namespace **QuantLib**

10.266 ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference

10.266.1 Detailed Description

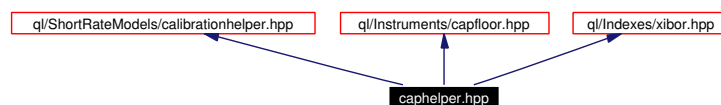
CapHelper calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for caphelper.hpp:



Namespaces

- namespace **QuantLib**

10.267 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference

10.267.1 Detailed Description

Swaption calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for swaptionhelper.hpp:



Namespaces

- namespace **QuantLib**

10.268 ql/ShortRateModels/model.hpp File Reference

10.268.1 Detailed Description

Abstract interest rate model class.

```
#include <ql/option.hpp>
```

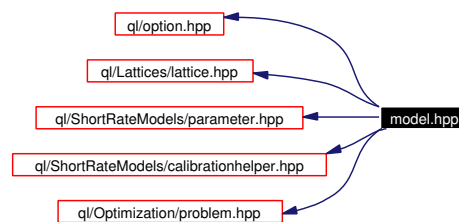
```
#include <ql/Lattices/lattice.hpp>
```

```
#include <ql/ShortRateModels/parameter.hpp>
```

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Optimization/problem.hpp>
```

Include dependency graph for model.hpp:



Namespaces

- namespace **QuantLib**

10.269 ql/ShortRateModels/onefactormodel.hpp File Reference

10.269.1 Detailed Description

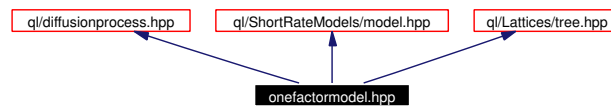
Abstract one-factor interest rate model class.

```
#include <ql/diffusionprocess.hpp>
```

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/Lattices/tree.hpp>
```

Include dependency graph for onefactormodel.hpp:



Namespaces

- namespace **QuantLib**

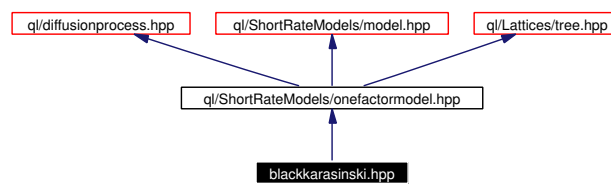
10.270 ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference

10.270.1 Detailed Description

Black-Karasinski model.

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for blackkarasinski.hpp:



Namespaces

- namespace **QuantLib**

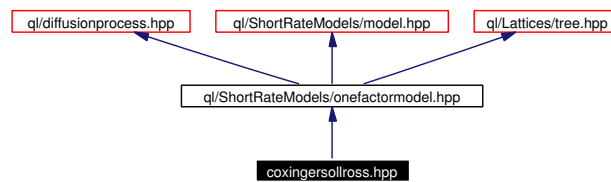
10.271 ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference

10.271.1 Detailed Description

Cox-Ingersoll-Ross model.

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for coxingersollross.hpp:



Namespaces

- namespace **QuantLib**

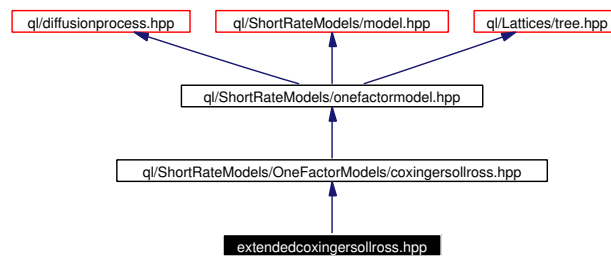
10.272 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference

10.272.1 Detailed Description

Extended Cox-Ingersoll-Ross model.

```
#include <ql/ShortRateModels/OneFactorModels/coxingersollross.hpp>
```

Include dependency graph for extendedcoxingersollross.hpp:



Namespaces

- namespace **QuantLib**

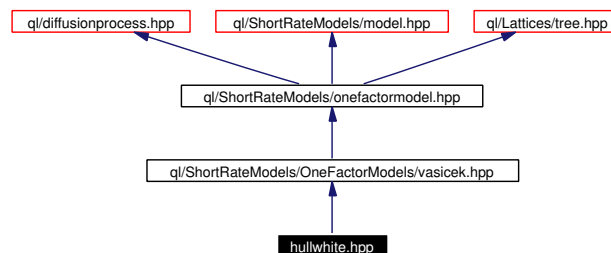
10.273 ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference

10.273.1 Detailed Description

Hull & White (HW) model.

```
#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>
```

Include dependency graph for hullwhite.hpp:



Namespaces

- namespace **QuantLib**

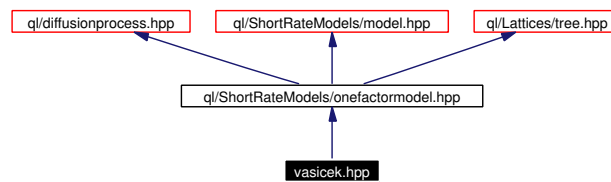
10.274 ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference

10.274.1 Detailed Description

Vasicek model class.

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for vasicek.hpp:



Namespaces

- namespace **QuantLib**

10.275 ql/ShortRateModels/parameter.hpp File Reference

10.275.1 Detailed Description

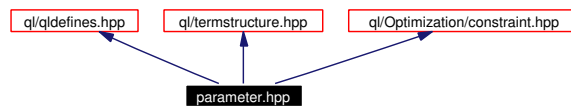
Model parameter classes.

```
#include <ql/qldefines.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <ql/Optimization/constraint.hpp>
```

Include dependency graph for parameter.hpp:



Namespaces

- namespace **QuantLib**

10.276 ql/ShortRateModels/twofactormodel.hpp File Reference

10.276.1 Detailed Description

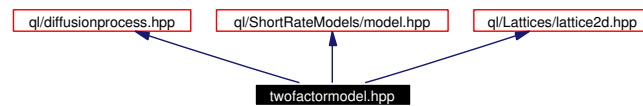
Abstract two-factor interest rate model class.

```
#include <ql/diffusionprocess.hpp>
```

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/Lattices/lattice2d.hpp>
```

Include dependency graph for twofactormodel.hpp:



Namespaces

- namespace **QuantLib**

10.277 ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference

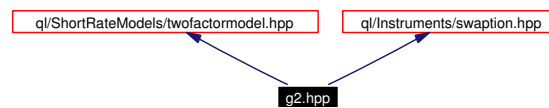
10.277.1 Detailed Description

Two-factor additive Gaussian Model G2++.

```
#include <ql/ShortRateModels/twofactormodel.hpp>
```

```
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for g2.hpp:



Namespaces

- namespace **QuantLib**

10.278 ql/solver1d.hpp File Reference

10.278.1 Detailed Description

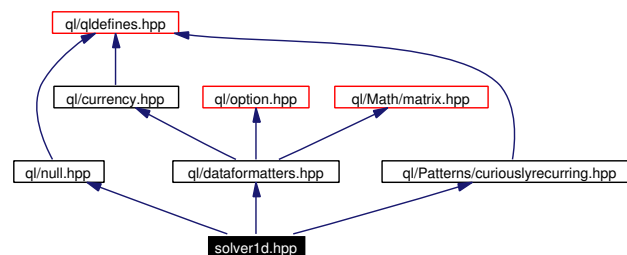
Abstract 1-D solver class.

```
#include <ql/null.hpp>
```

```
#include <ql/dataformatters.hpp>
```

```
#include <ql/Patterns/curiouslyrecurring.hpp>
```

Include dependency graph for solver1d.hpp:



Namespaces

- namespace **QuantLib**

Defines

- `#define MAX_FUNCTION_EVALUATIONS 100`

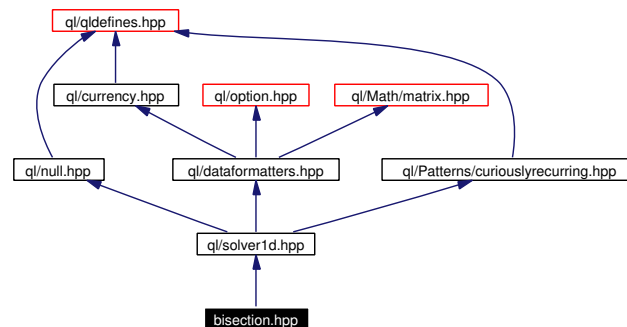
10.279 ql/Solvers1D/bisection.hpp File Reference

10.279.1 Detailed Description

bisection 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for bisection.hpp:



Namespaces

- namespace **QuantLib**

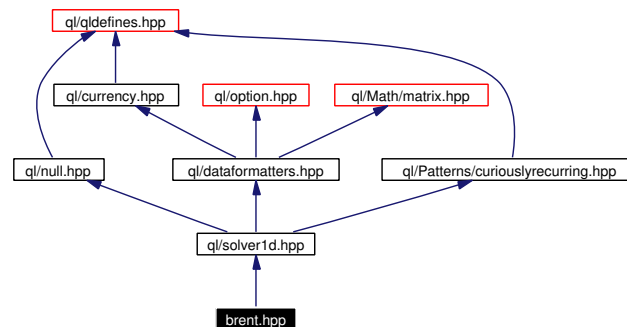
10.280 ql/Solvers1D/brent.hpp File Reference

10.280.1 Detailed Description

Brent 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for brent.hpp:



Namespaces

- namespace **QuantLib**

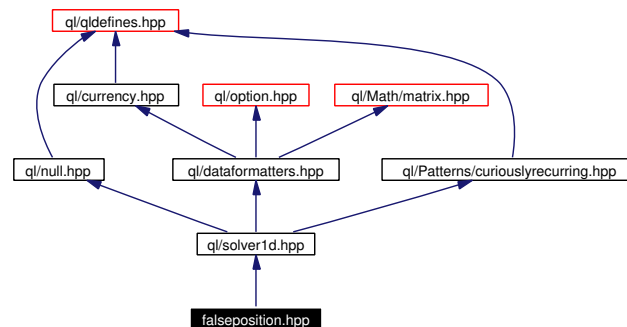
10.281 ql/Solvers1D/falseposition.hpp File Reference

10.281.1 Detailed Description

false-position 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for falseposition.hpp:



Namespaces

- namespace **QuantLib**

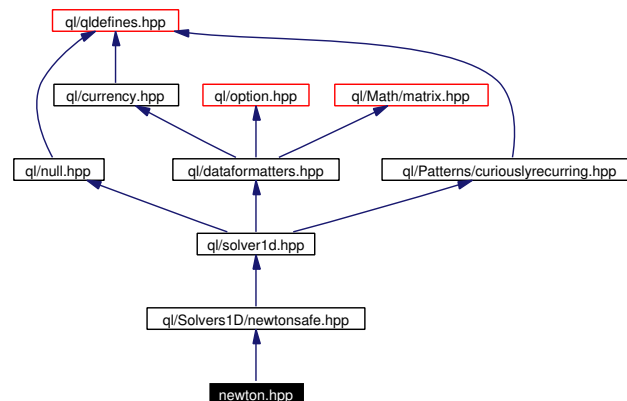
10.282 ql/Solvers1D/newton.hpp File Reference

10.282.1 Detailed Description

Newton 1-D solver.

```
#include <ql/Solvers1D/newtonsafe.hpp>
```

Include dependency graph for newton.hpp:



Namespaces

- namespace **QuantLib**

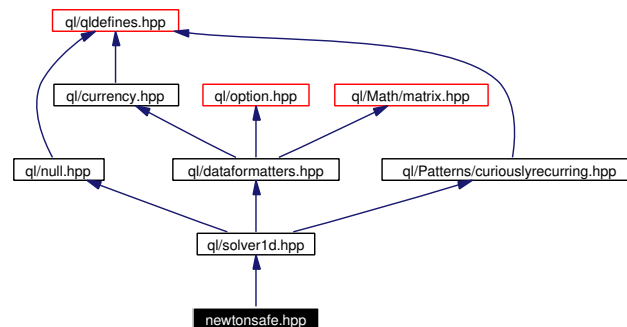
10.283 ql/Solvers1D/newtonsafe.hpp File Reference

10.283.1 Detailed Description

Safe (bracketed) Newton 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for newtonsafe.hpp:



Namespaces

- namespace **QuantLib**

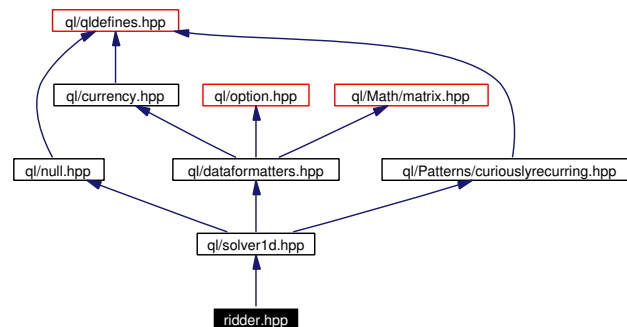
10.284 ql/Solvers1D/ridder.hpp File Reference

10.284.1 Detailed Description

Ridder 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for ridder.hpp:



Namespaces

- namespace **QuantLib**

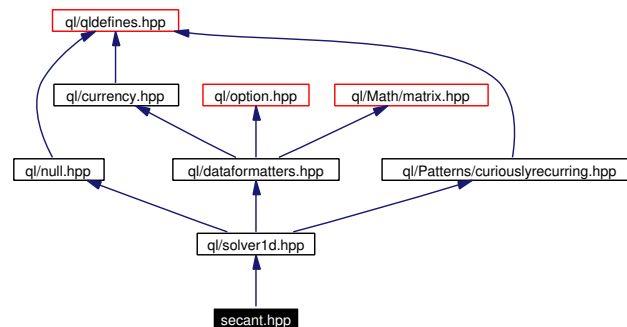
10.285 ql/Solvers1D/secant.hpp File Reference

10.285.1 Detailed Description

secant 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for secant.hpp:



Namespaces

- namespace **QuantLib**

10.286 ql/stochasticprocess.hpp File Reference

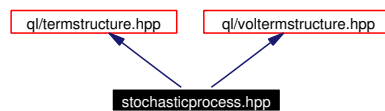
10.286.1 Detailed Description

Base stochastic process class.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for stochasticprocess.hpp:



Namespaces

- namespace **QuantLib**

10.287 ql/swapvolstructure.hpp File Reference

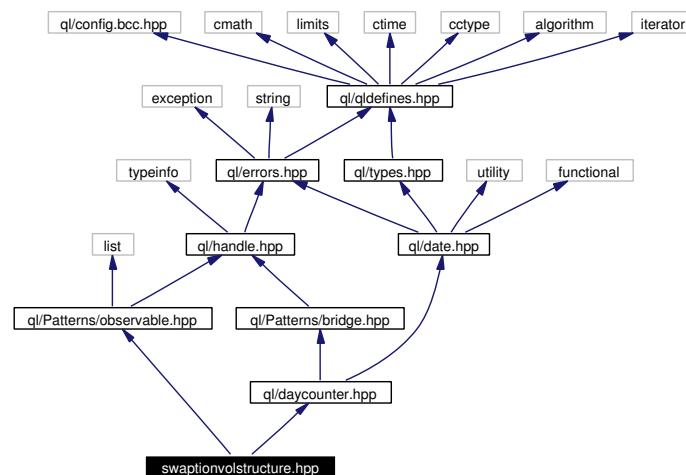
10.287.1 Detailed Description

Swap option volatility structure.

```
#include <ql/daycounter.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for swapvolstructure.hpp:



Namespaces

- namespace **QuantLib**

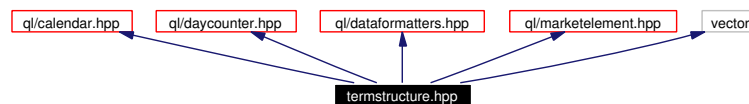
10.288 ql/termstructure.hpp File Reference

10.288.1 Detailed Description

Term structure.

```
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/dataformatters.hpp>
#include <ql/marketelement.hpp>
#include <vector>
```

Include dependency graph for termstructure.hpp:



Namespaces

- namespace **QuantLib**

10.289 ql/TermStructures/affinetermstructure.hpp File Reference

10.289.1 Detailed Description

Affine term structure.

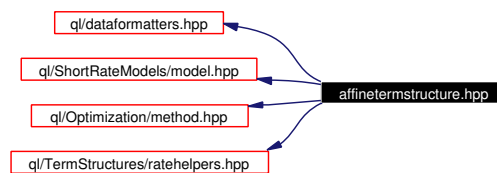
```
#include <ql/dataformatters.hpp>
```

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/Optimization/method.hpp>
```

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Include dependency graph for affinetermstructure.hpp:



Namespaces

- namespace **QuantLib**

10.290 ql/TermStructures/compoundforward.hpp File Reference

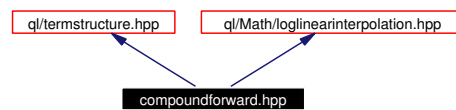
10.290.1 Detailed Description

compounded forward term structure

```
#include <ql/termstructure.hpp>
```

```
#include <ql/Math/loglinearinterpolation.hpp>
```

Include dependency graph for compoundforward.hpp:



Namespaces

- namespace **QuantLib**

10.291 ql/TermStructures/discountcurve.hpp File Reference

10.291.1 Detailed Description

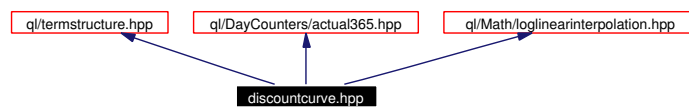
pre-bootstrapped discount factor structure

```
#include <ql/termstructure.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

```
#include <ql/Math/loglinearinterpolation.hpp>
```

Include dependency graph for discountcurve.hpp:



Namespaces

- namespace **QuantLib**

10.292 ql/TermStructures/drifftermstructure.hpp File Reference

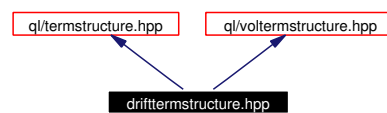
10.292.1 Detailed Description

Drift term structure.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for drifftermstructure.hpp:



Namespaces

- namespace **QuantLib**

10.293 ql/TermStructures/extendeddiscountcurve.hpp File Reference

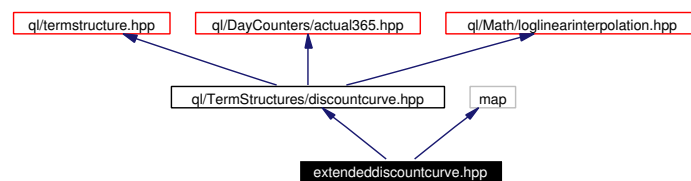
10.293.1 Detailed Description

discount factor structure with detailed compound-forward calculation

```
#include <ql/TermStructures/discountcurve.hpp>
```

```
#include <map>
```

Include dependency graph for extendeddiscountcurve.hpp:



Namespaces

- namespace **QuantLib**

10.294 ql/TermStructures/flatforward.hpp File Reference

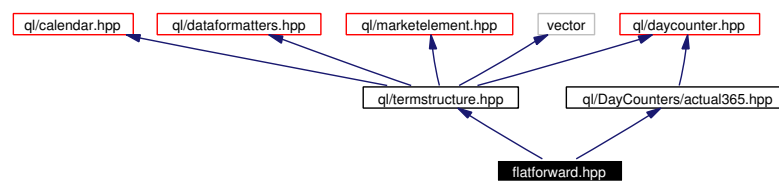
10.294.1 Detailed Description

flat forward rate term structure

```
#include <ql/termstructure.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

Include dependency graph for flatforward.hpp:



Namespaces

- namespace **QuantLib**

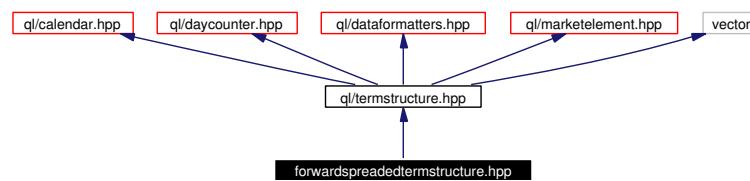
10.295 ql/TermStructures/forwardspreadedtermstructure.hpp File Reference

10.295.1 Detailed Description

Forward spreaded term structure.

```
#include <ql/termstructure.hpp>
```

Include dependency graph for forwardspreadedtermstructure.hpp:



Namespaces

- namespace **QuantLib**

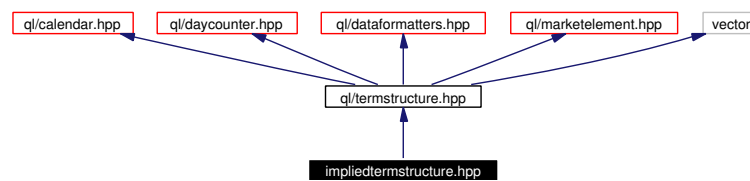
10.296 ql/TermStructures/IMPLIEDTERMSTRUCTURE.hpp File Reference

10.296.1 Detailed Description

Implied term structure.

```
#include <ql/termstructure.hpp>
```

Include dependency graph for IMPLIEDTERMSTRUCTURE.hpp:



Namespaces

- namespace **QuantLib**

10.297 ql/TermStructures/piecewiseflatforward.hpp File Reference

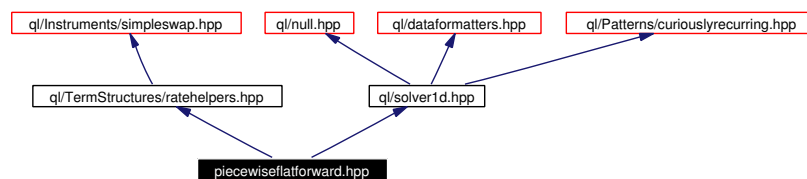
10.297.1 Detailed Description

piecewise flat forward term structure

```
#include <ql/TermStructures/ratehelpers.hpp>
```

```
#include <ql/solver1d.hpp>
```

Include dependency graph for piecewiseflatforward.hpp:



Namespaces

- namespace **QuantLib**

10.298 ql/TermStructures/quantotermstructure.hpp File Reference

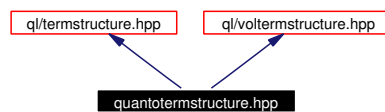
10.298.1 Detailed Description

Quanto term structure.

```
#include <ql/termstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for quantotermstructure.hpp:



Namespaces

- namespace **QuantLib**

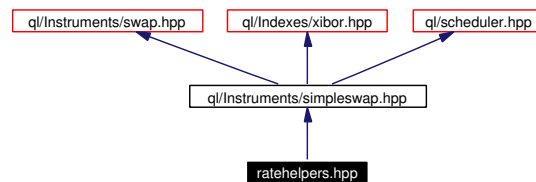
10.299 ql/TermStructures/ratehelpers.hpp File Reference

10.299.1 Detailed Description

rate helpers base class

```
#include <ql/Instruments/simpleswap.hpp>
```

Include dependency graph for ratehelpers.hpp:



Namespaces

- namespace **QuantLib**

10.300 ql/TermStructures/zerocurve.hpp File Reference

10.300.1 Detailed Description

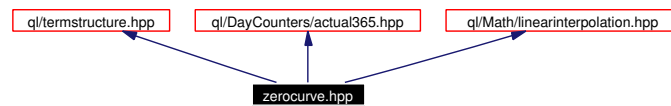
pre-bootstrapped zero curve structure

```
#include <ql/termstructure.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

```
#include <ql/Math/linearinterpolation.hpp>
```

Include dependency graph for zerocurve.hpp:



Namespaces

- namespace **QuantLib**

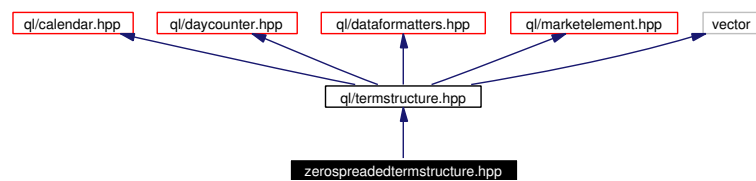
10.301 ql/TermStructures/zerospreadedtermstructure.hpp File Reference

10.301.1 Detailed Description

Zero spreaded term structure.

```
#include <ql/termstructure.hpp>
```

Include dependency graph for zerospreadedtermstructure.hpp:



Namespaces

- namespace **QuantLib**

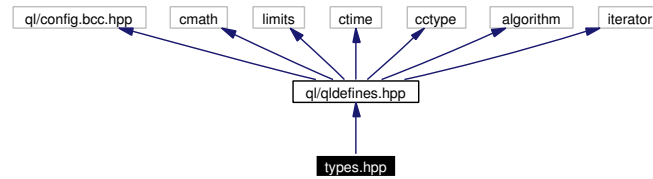
10.302 ql/types.hpp File Reference

10.302.1 Detailed Description

Custom types.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for types.hpp:



Namespaces

- namespace **QuantLib**

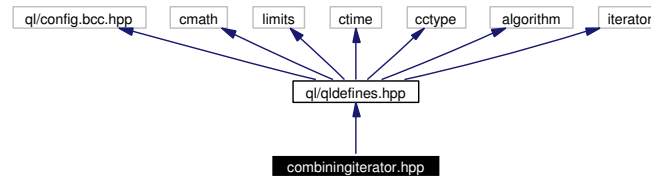
10.303 ql/Utilities/combiningiterator.hpp File Reference

10.303.1 Detailed Description

Iterator mapping a function to a set of underlying sequences.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for combiningiterator.hpp:



Namespaces

- namespace **QuantLib**

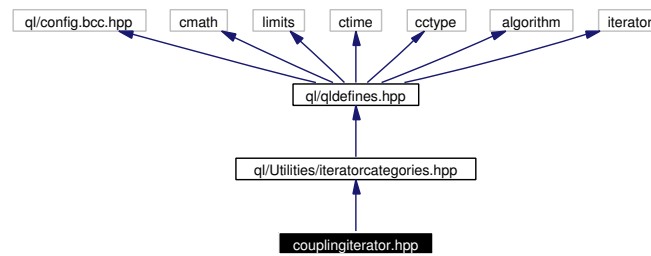
10.304 ql/Utilities/couplingiterator.hpp File Reference

10.304.1 Detailed Description

Iterator mapping a function to a pair of underlying sequences.

```
#include <ql/Utilities/iteratorcategories.hpp>
```

Include dependency graph for couplingiterator.hpp:



Namespaces

- namespace **QuantLib**

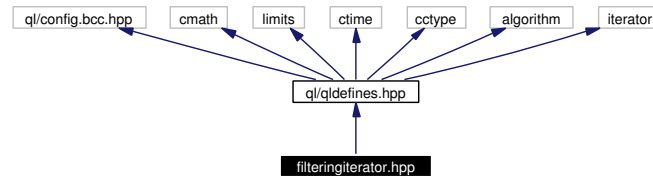
10.305 ql/Utilities/filteringiterator.hpp File Reference

10.305.1 Detailed Description

Iterator filtering undesired data.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for filteringiterator.hpp:



Namespaces

- namespace **QuantLib**

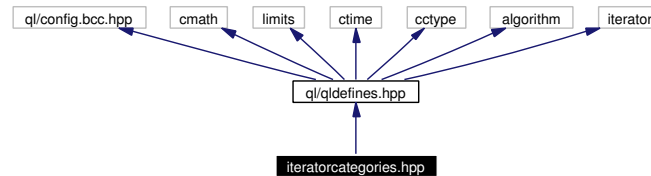
10.306 ql/Utilities/iteratorcategories.hpp File Reference

10.306.1 Detailed Description

Lowest common denominator between two iterator categories.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for iteratorcategories.hpp:



Namespaces

- namespace **QuantLib**

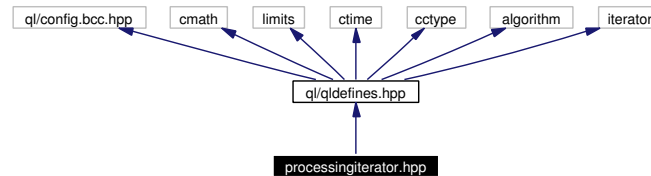
10.307 ql/Utilities/processingiterator.hpp File Reference

10.307.1 Detailed Description

Iterator mapping a unary function to an underlying sequence.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for processingiterator.hpp:



Namespaces

- namespace **QuantLib**

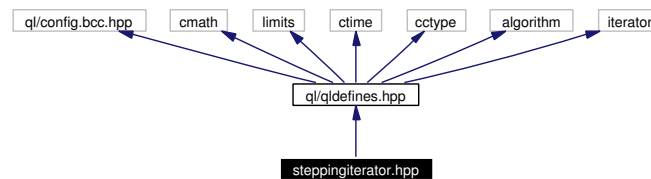
10.308 ql/Utilities/steppingiterator.hpp File Reference

10.308.1 Detailed Description

Iterator advancing in constant steps.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for steppingiterator.hpp:



Namespaces

- namespace **QuantLib**

10.309 ql/Volatilities/blackconstantvol.hpp File Reference

10.309.1 Detailed Description

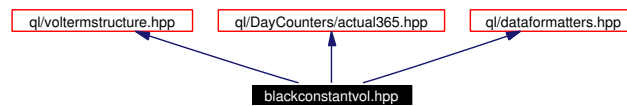
Black constant volatility, no time dependence, no strike dependence.

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

```
#include <ql/dataformatters.hpp>
```

Include dependency graph for blackconstantvol.hpp:



Namespaces

- namespace **QuantLib**

10.310 ql/Volatilities/blackvariancecurve.hpp File Reference

10.310.1 Detailed Description

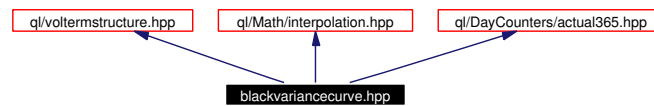
Black volatility curve modelled as variance curve.

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/Math/interpolation.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

Include dependency graph for blackvariancecurve.hpp:



Namespaces

- namespace **QuantLib**

10.311 ql/Volatilities/blackvariancesurface.hpp File Reference

10.311.1 Detailed Description

Black volatility surface modelled as variance surface.

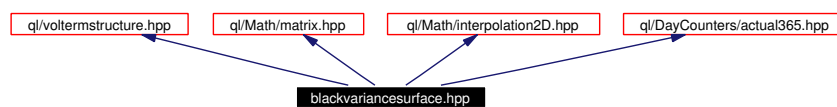
```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Math/interpolation2D.hpp>
```

```
#include <ql/DayCounters/actual365.hpp>
```

Include dependency graph for blackvariancesurface.hpp:



Namespaces

- namespace **QuantLib**

10.312 ql/Volatilities/capflatvolvector.hpp File Reference

10.312.1 Detailed Description

Cap/floor at-the-money flat volatility vector.

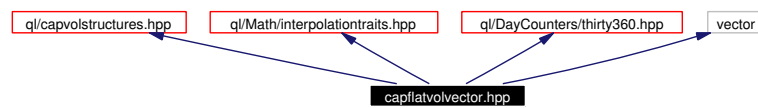
```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/Math/interpolationtraits.hpp>
```

```
#include <ql/DayCounters/thirty360.hpp>
```

```
#include <vector>
```

Include dependency graph for capflatvolvector.hpp:



Namespaces

- namespace **QuantLib**

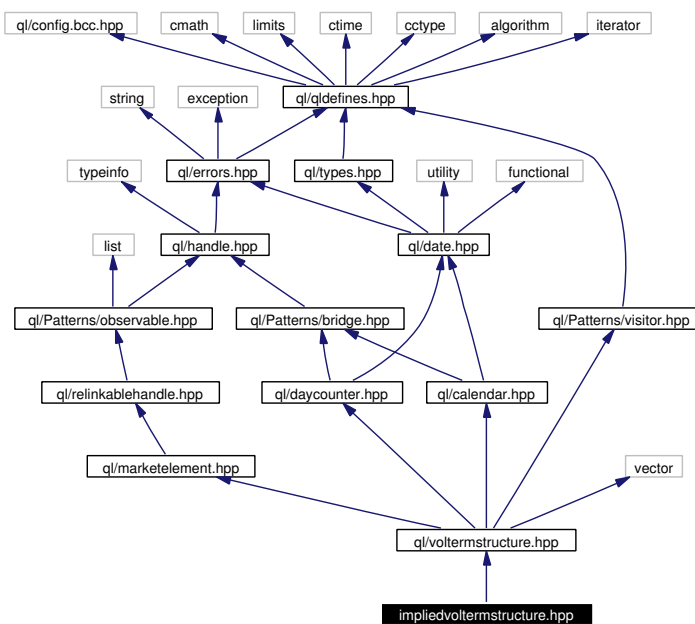
10.313 ql/Volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp File Reference

10.313.1 Detailed Description

Implied Black Vol Term Structure.

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for IMPLIEDVOLTERMSTRUCTURE.hpp:



Namespaces

- namespace **QuantLib**

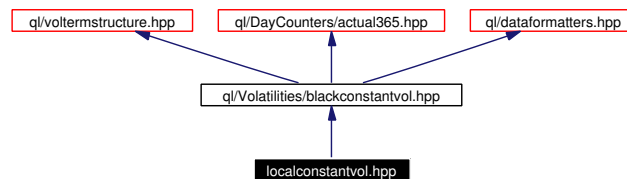
10.314 ql/Volatilities/localconstantvol.hpp File Reference

10.314.1 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**

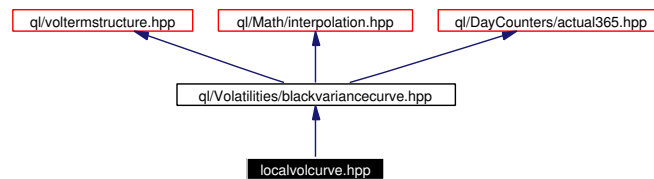
10.315 ql/Volatilities/localvolcurve.hpp File Reference

10.315.1 Detailed Description

Local volatility curve derived from a Black curve.

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for localvolcurve.hpp:



Namespaces

- namespace **QuantLib**

10.316 ql/Volatilities/localvolsurface.hpp File Reference

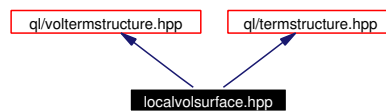
10.316.1 Detailed Description

Local volatility surface derived from a Black vol surface.

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/termstructure.hpp>
```

Include dependency graph for localvolsurface.hpp:



Namespaces

- namespace **QuantLib**

10.317 ql/Volatilities/swaptionvolmatrix.hpp File Reference

10.317.1 Detailed Description

Swaption at-the-money volatility matrix.

```
#include <ql/swaptionvolstructure.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Math/interpolationtraits.hpp>
```

```
#include <vector>
```

Include dependency graph for swaptionvolmatrix.hpp:



Namespaces

- namespace **QuantLib**

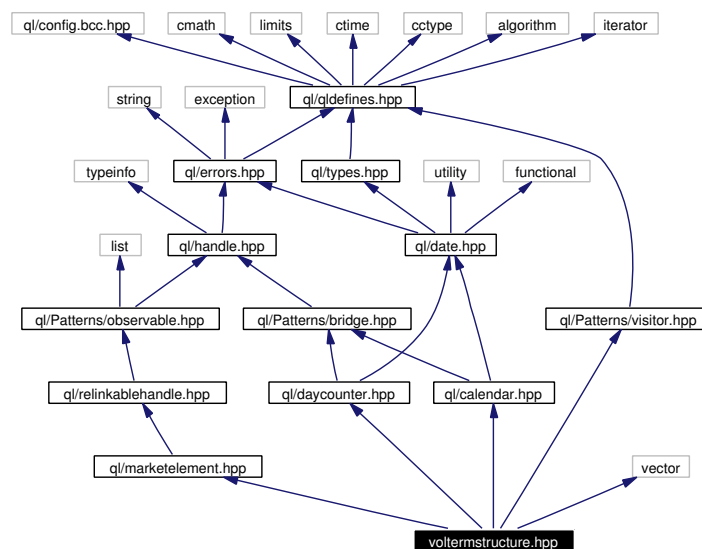
10.318 ql/voltermstructure.hpp File Reference

10.318.1 Detailed Description

Volatility term structures.

```
#include <ql/calendar.hpp>
#include <ql/daycounter.hpp>
#include <ql/marketelement.hpp>
#include <ql/Patterns/visitor.hpp>
#include <vector>
```

Include dependency graph for voltermstructure.hpp:



Namespaces

- namespace **QuantLib**

Chapter 11

QuantLib Example Documentation

11.1 AmericanOption.cpp

This example calculates American options using different methods

```
#include <ql/quantlib.hpp>

using namespace QuantLib;

int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        std::cout << "Using " << QL_VERSION << std::endl << std::endl;

        // our option
        Option::Type type(Option::Put);
        double underlying = 36;
        double strike = 40;
        Spread dividendYield = 0.00;
        Rate riskFreeRate = 0.06;
        double volatility = 0.20;

        Date todaysDate(15, May, 1998);
        Date settlementDate(17, May, 1998);
        Date exerciseDate(17, May, 1999);
        DayCounter rateDayCounter = Actual365();
        Time maturity = rateDayCounter.yearFraction(settlementDate,
                                                    exerciseDate);

        std::cout << "option type = " << OptionTypeFormatter::toString(type)
                  << std::endl;
        std::cout << "Time to maturity = " << maturity
                  << std::endl;
        std::cout << "Underlying price = " << underlying
                  << std::endl;
        std::cout << "Strike = " << strike
                  << std::endl;
        std::cout << "Risk-free interest rate = " << riskFreeRate
                  << std::endl;
        std::cout << "dividend yield = " << dividendYield
                  << std::endl;
        std::cout << "Volatility = " << volatility
                  << std::endl;
```

```

std::cout << std::endl;

std::string method;

double value, discrepancy, rightValue, relativeDiscrepancy;
rightValue = (type == Option::Put ? 4.48667344 : 2.17372645);

std::cout << std::endl ;

// write column headings
std::cout << "Method\t\t\t\t Value\t\tDiscrepancy"
            "\tRel. Discr." << std::endl;

Date midlifeDate(19, November, 1998);
std::vector<Date> exDates(2);
exDates[0]=midlifeDate;
exDates[1]=exerciseDate;

Handle<Exercise> exercise(new EuropeanExercise(exerciseDate));
Handle<Exercise> amExercise(new AmericanExercise(settlementDate,
                                                exerciseDate));
Handle<Exercise> berExercise(new BermudanExercise(exDates));

RelinkableHandle<Quote> underlyingH(
    Handle<Quote>(new SimpleQuote(underlying)));

// bootstrap the yield/dividend/vol curves
RelinkableHandle<TermStructure> flatTermStructure(
    Handle<TermStructure>(
        new FlatForward(todaysDate, settlementDate,
                        riskFreeRate, rateDayCounter)));
RelinkableHandle<TermStructure> flatDividendTS(
    Handle<TermStructure>(
        new FlatForward(todaysDate, settlementDate,
                        dividendYield, rateDayCounter)));
RelinkableHandle<BlackVolTermStructure> flatVolTS(
    Handle<BlackVolTermStructure>(
        new BlackConstantVol(settlementDate, volatility)));

std::vector<Date> dates(4);
dates[0] = settlementDate.plusMonths(1);
dates[1] = exerciseDate;
dates[2] = exerciseDate.plusMonths(6);
dates[3] = exerciseDate.plusMonths(12);
std::vector<double> strikes(4);
strikes[0] = underlying*0.9;
strikes[1] = underlying;
strikes[2] = underlying*1.1;
strikes[3] = underlying*1.2;

Matrix vols(4,4);
vols[0][0] = volatility*1.1; vols[0][1] = volatility;
    vols[0][2] = volatility*0.9; vols[0][3] = volatility*0.8;
vols[1][0] = volatility*1.1; vols[1][1] = volatility;
    vols[1][2] = volatility*0.9; vols[1][3] = volatility*0.8;
vols[2][0] = volatility*1.1; vols[2][1] = volatility;
    vols[2][2] = volatility*0.9; vols[2][3] = volatility*0.8;
vols[3][0] = volatility*1.1; vols[3][1] = volatility;
    vols[3][2] = volatility*0.9; vols[3][3] = volatility*0.8;
RelinkableHandle<BlackVolTermStructure> blackSurface(
    Handle<BlackVolTermStructure>(
        new BlackVarianceSurface(settlementDate, dates, strikes, vols)));

Handle<StrikedTypePayoff> payoff(new
    PlainVanillaPayoff(type, strike));

```

```

Handle<BlackScholesStochasticProcess> stochasticProcess(new
    BlackScholesStochasticProcess(
        underlyingH,
        flatDividendTS,
        flatTermStructure,
        flatVolTS));

// European option
VanillaOption euroOption(stochasticProcess, payoff, exercise,
    Handle<PricingEngine>(new AnalyticEuropeanEngine()));

// method: Black Scholes Engine
method = "equivalent european option      ";
value = euroOption.NPV();
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << "N/A\t\t"
    << "N/A\t\t"
    << std::endl;

// American option
VanillaOption option(stochasticProcess, payoff, amExercise);

Size timeSteps = 801;

// Binomial Method (JR)
method = "Binomial Jarrow-Rudd            ";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Binomial Method (CRR)
method = "Binomial Cox-Ross-Rubinstein    ";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Equal Probability Additive Binomial Tree (EQP)
method = "Additive Equiprobabilities      ";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<AdditiveEQPBinoialTree>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Equal Jumps Additive Binomial Tree (Trigeorgis)
method = "Binomial Trigeorgis             ";
option.setPricingEngine(Handle<PricingEngine>(

```

```

        new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Tian Binomial Tree (third moment matching)
method = "Binomial Tian ";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<Tian>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Leisen-Reimer Binomial Tree
method = "Binomial Leisen-Reimer ";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Barone-Adesi and Whaley approximation
method = "Barone-Adesi and Whaley approx. ";
option.setPricingEngine(Handle<PricingEngine>(
    new BaroneAdesiWhaleyApproximationEngine));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Bjerksund and Stensland approximation
method = "Bjerksund and Stensland approx. ";
option.setPricingEngine(Handle<PricingEngine>(
    new BjerksundStenslandApproximationEngine));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DoubleFormatter::toString(value, 6) << "\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

return 0;
} catch (std::exception& e) {
    std::cout << e.what() << std::endl;
    return 1;
} catch (...) {

```

```
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

11.2 BermudanSwaption.cpp

This is an example of using the QuantLib short rate models.

```
#include <ql/quantlib.hpp>

using namespace QuantLib;

//Number of swaptions to be calibrated to...

Size numRows = 5;
Size numCols = 10;

unsigned int swaptionLengths[] = {1,2,3,4,5,7,10,15,20,25,30};
double swaptionVols[] = {
    23.92, 22.80, 19.8, 18.1, 16.0, 14.26, 13.56, 12.79, 12.3, 11.09,
    21.85, 21.50, 19.5, 17.2, 14.7, 13.23, 12.59, 12.29, 11.1, 10.30,
    19.46, 19.40, 17.9, 15.9, 13.9, 12.69, 12.15, 11.83, 10.8, 10.00,
    17.97, 17.80, 16.7, 14.9, 13.4, 12.28, 11.89, 11.48, 10.5, 9.80,
    16.29, 16.40, 15.1, 14.0, 12.9, 12.01, 11.46, 11.08, 10.4, 9.77,
    14.71, 14.90, 14.3, 13.2, 12.3, 11.49, 11.12, 10.70, 10.1, 9.57,
    12.93, 13.30, 12.8, 12.2, 11.6, 10.82, 10.47, 10.21, 9.8, 9.51,
    12.70, 12.10, 11.9, 11.2, 10.8, 10.40, 10.20, 10.00, 9.5, 9.00,
    12.30, 11.60, 11.6, 10.9, 10.5, 10.30, 10.00, 9.80, 9.3, 8.80,
    12.00, 11.40, 11.5, 10.8, 10.3, 10.00, 9.80, 9.60, 9.5, 9.10,
    11.50, 11.20, 11.3, 10.6, 10.2, 10.10, 9.70, 9.50, 9.4, 8.60};

void calibrateModel(const Handle<ShortRateModel>& model,
                    CalibrationSet& calibs,
                    double lambda) {

    Simplex om(lambda, 1e-9);
    om.setEndCriteria(EndCriteria(10000, 1e-7));
    model->calibrate(calibs, om);

    #if defined(QL_PATCH_DARWIN)
    // to be investigated
    return;
    #endif

    //Output the implied Black volatilities
    Size i;
    for (i=0; i<numRows; i++) {
        std::cout << IntegerFormatter::toString(swaptionLengths[i],2) << "y|";
        for (Size j=0; j<numCols; j++) {
            Size k = i*numCols + j;
            double npv = calibs[k]->modelValue();
            double implied = calibs[k]->impliedVolatility(npv, 1e-4,
                1000, 0.05, 0.50)*100.0;
            std::cout << DoubleFormatter::toString(implied,1,4) << " (";
            k = i*10 + j;
            double diff = implied - swaptionVols[k];
            std::cout << DoubleFormatter::toString(diff,1,4)
                << ")|";
        }
        std::cout << std::endl;
    }
}

int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        Date todaysDate(15, February, 2002);
        Calendar calendar = TARGET();
```

```

// Date settlementDate = calendar.advance(todaysDate,
//                                         settlementDays, Days);
Date settlementDate(19, February, 2002);

//Instruments used to bootstrap the yield curve:
std::vector<Handle<RateHelper> > instruments;

//Deposit rates
DayCounter depositDayCounter = Thirty360();
int settlementDays = 2;

Rate weekRates[3] = {3.295, 3.3, 3.3};
Size i;
for (i=0; i<3; i++) {
    RelinkableHandle<Quote> depositRate(
        Handle<Quote>(new SimpleQuote(weekRates[i]*0.01)));
    Handle<RateHelper> depositHelper(new DepositRateHelper(
        depositRate, i+1, Weeks, settlementDays, calendar,
        ModifiedFollowing, depositDayCounter));
    instruments.push_back(depositHelper);
}

Rate depositRates[12] = {
    3.31, 3.32, 3.29, 3.35, 3.315, 3.33,
    3.395, 3.41, 3.41, 3.49, 3.54, 3.53};

for (i=0; i<11; i++) {
    RelinkableHandle<Quote> depositRate(
        Handle<Quote>(new SimpleQuote(depositRates[i]*0.01)));
    Handle<RateHelper> depositHelper(new DepositRateHelper(
        depositRate, i+1, Months, settlementDays, calendar,
        ModifiedFollowing, depositDayCounter));
    instruments.push_back(depositHelper);
}

//Swap rates
Rate swapRates[13] = {
    3.6425, 4.0875, 4.38, 4.5815, 4.74325, 4.87375,
    4.9775, 5.07, 5.13, 5.1825, 5.36, 5.45125, 5.43875};
int swapYears[13] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 15, 20, 30};

int swFixedLegFrequency = 1;
bool swFixedLegIsAdjusted = false;
DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
int swFloatingLegFrequency = 2;

for (i=0; i<13; i++) {
    Handle<Quote> swapRate(new SimpleQuote(swapRates[i]*0.01));
    Handle<RateHelper> swapHelper(new SwapRateHelper(
        RelinkableHandle<Quote>(swapRate),
        swapYears[i], Years, settlementDays,
        calendar, ModifiedFollowing,
        swFixedLegFrequency,
        swFixedLegIsAdjusted, swFixedLegDayCounter,
        swFloatingLegFrequency));
    instruments.push_back(swapHelper);
}

// bootstrapping the yield curve
Handle<PiecewiseFlatForward> myTermStructure(new
    PiecewiseFlatForward(todaysDate, settlementDate, instruments,
        depositDayCounter));

RelinkableHandle<TermStructure> rhTermStructure;
rhTermStructure.linkTo(myTermStructure);

```

```

//Define the ATM/OTM/ITM swaps
int fixedLegFrequency = 1;
bool fixedLegIsAdjusted = false;
RollingConvention roll = ModifiedFollowing;
DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
int floatingLegFrequency = 2;
bool payFixedRate = true;
int fixingDays = 2;
Rate dummyFixedRate = 0.03;
Handle<Xibor> indexSixMonths(new Euribor(6, Months, rhTermStructure));

Handle<SimpleSwap> swap(new SimpleSwap(
    payFixedRate, settlementDate.plusYears(1), 5, Years,
    calendar, roll, 1000.0, fixedLegFrequency, dummyFixedRate,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    indexSixMonths, fixingDays, 0.0, rhTermStructure));
Rate fixedATMRate = swap->fairRate();

Handle<SimpleSwap> atmSwap(new SimpleSwap(
    payFixedRate, settlementDate.plusYears(1), 5, Years,
    calendar, roll, 1000.0, fixedLegFrequency, fixedATMRate,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    indexSixMonths, fixingDays, 0.0, rhTermStructure));
Handle<SimpleSwap> otmSwap(new SimpleSwap(
    payFixedRate, settlementDate.plusYears(1), 5, Years,
    calendar, roll, 1000.0, fixedLegFrequency, fixedATMRate * 1.2,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    indexSixMonths, fixingDays, 0.0, rhTermStructure));
Handle<SimpleSwap> itmSwap(new SimpleSwap(
    payFixedRate, settlementDate.plusYears(1), 5, Years,
    calendar, roll, 1000.0, fixedLegFrequency, fixedATMRate * 0.8,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    indexSixMonths, fixingDays, 0.0, rhTermStructure));

std::vector<Period> swaptionMaturities;
swaptionMaturities.push_back(Period(1, Months));
swaptionMaturities.push_back(Period(3, Months));
swaptionMaturities.push_back(Period(6, Months));
swaptionMaturities.push_back(Period(1, Years));
swaptionMaturities.push_back(Period(2, Years));
swaptionMaturities.push_back(Period(3, Years));
swaptionMaturities.push_back(Period(4, Years));
swaptionMaturities.push_back(Period(5, Years));
swaptionMaturities.push_back(Period(7, Years));
swaptionMaturities.push_back(Period(10, Years));

CalibrationSet swaptions;

//List of times that have to be included in the timegrid
std::list<Time> times;

for (i=0; i<numRows; i++) {
    for (unsigned int j=0; j<numCols; j++) {
        unsigned int k = i*10 + j;
        Handle<Quote> vol(new SimpleQuote(swaptionVols[k]*0.01));
        swaptions.push_back(Handle<CalibrationHelper>(
            new SwaptionHelper(swaptionMaturities[j],
                               Period(swaptionLengths[i], Years),
                               RelinkableHandle<Quote>(vol),
                               indexSixMonths,
                               rhTermStructure)));
        swaptions.back()->addTimesTo(times);
    }
}
const std::vector<Time> termTimes = myTermStructure->times();
for (i=0; i<termTimes.size(); i++)

```

```

        times.push_back(termTimes[i]);
// Building time-grid
TimeGrid grid(times.begin(), times.end(), 30);

Handle<HullWhite> modelHW(new HullWhite(rhTermStructure));
Handle<HullWhite> modelHW2(new HullWhite(rhTermStructure));
Handle<BlackKarasinski> modelBK(new BlackKarasinski(rhTermStructure));

std::cout << "Calibrating to swaptions" << std::endl;

std::cout << "Hull-White (analytic formulae):" << std::endl;
swaptions.setPricingEngine(
    Handle<PricingEngine>(new JamshidianSwaption(modelHW)));

calibrateModel(modelHW, swaptions, 0.05);
std::cout << "calibrated to "
            << modelHW->params()
            << std::endl
            << std::endl;

std::cout << "Hull-White (numerical calibration):" << std::endl;
swaptions.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelHW2, grid)));

calibrateModel(modelHW2, swaptions, 0.05);
std::cout << "calibrated to "
            << modelHW2->params()
            << std::endl
            << std::endl;

std::cout << "Black-Karasinski: " << std::endl;
swaptions.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelBK, grid)));
calibrateModel(modelBK, swaptions, 0.05);
std::cout << "calibrated to "
            << modelBK->params()
            << std::endl
            << std::endl;

std::cout << "Pricing an ATM bermudan swaption" << std::endl;

//Define the bermudan swaption
std::vector<Date> bermudanDates;
const std::vector<Handle<CashFlow> >& leg = swap->floatingLeg();
for (i=0; i<leg.size(); i++) {
    #if defined(HAVE_BOOST)
        Handle<Coupon> coupon =
            boost::dynamic_pointer_cast<Coupon>(leg[i]);
    #else
        Handle<Coupon> coupon = leg[i];
    #endif
    bermudanDates.push_back(coupon->accrualStartDate());
}

Handle<Exercise> bermudaExercise(new BermudanExercise(
    bermudanDates));

Swaption bermudanSwaption(atmSwap,
    bermudaExercise,
    rhTermStructure,
    Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));

//Do the pricing for each model
bermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));

```

```

std::cout << "HW:          " << bermudanSwaption.NPV() << std::endl;

bermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelHW2, 100)));
std::cout << "HW (num): " << bermudanSwaption.NPV() << std::endl;

bermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelBK, 100)));
std::cout << "BK:          " << bermudanSwaption.NPV() << std::endl;

std::cout << "Pricing an OTM bermudan swaption" << std::endl;

Swaption otmBermudanSwaption(otmSwap,
    bermudaExercise,
    rhTermStructure,
    Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));

//Do the pricing for each model
otmBermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));
std::cout << "HW:          " << otmBermudanSwaption.NPV() << std::endl;

otmBermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelHW2, 100)));
std::cout << "HW (num): " << otmBermudanSwaption.NPV() << std::endl;

otmBermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelBK, 100)));
std::cout << "BK:          " << otmBermudanSwaption.NPV() << std::endl;

std::cout << "Pricing an ITM bermudan swaption" << std::endl;

Swaption itmBermudanSwaption(itmSwap,
    bermudaExercise,
    rhTermStructure,
    Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));

//Do the pricing for each model
itmBermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelHW, 100)));
std::cout << "HW:          " << itmBermudanSwaption.NPV() << std::endl;
itmBermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelHW2, 100)));
std::cout << "HW (num): " << itmBermudanSwaption.NPV() << std::endl;
itmBermudanSwaption.setPricingEngine(
    Handle<PricingEngine>(new TreeSwaption(modelBK, 100)));
std::cout << "BK:          " << itmBermudanSwaption.NPV() << std::endl;

    return 0;
} catch (std::exception& e) {
    std::cout << e.what() << std::endl;
    return 1;
} catch (...) {
    std::cout << "unknown error" << std::endl;
    return 1;
}
}

```

11.3 DiscreteHedging.cpp

This is an example of using the QuantLib Monte Carlo framework.

```
/* This example computes profit and loss of a discrete interval hedging
   strategy and compares with the results of Derman & Kamal's (Goldman Sachs
   Equity Derivatives Research) Research Note: "When You Cannot Hedge
   Continuously: The Corrections to Black-Scholes"
   (http://www.gs.com/qs/doc/when\_you\_cannot\_hedge.pdf)
```

Suppose an option hedger sells an European option and receives the Black-Scholes value as the options premium. Then he follows a Black-Scholes hedging strategy, rehedging at discrete, evenly spaced time intervals as the underlying stock changes. At expiration, the hedger delivers the option payoff to the option holder, and unwinds the hedge. We are interested in understanding the final profit or loss of this strategy.

If the hedger had followed the exact Black-Scholes replication strategy, re-hedging continuously as the underlying stock evolved towards its final value at expiration, then, no matter what path the stock took, the final P&L would be exactly zero. When the replication strategy deviates from the exact Black-Scholes method, the final P&L may deviate from zero. This deviation is called the replication error. When the hedger rebalances at discrete rather than continuous intervals, the hedge is imperfect and the replication is inexact. The more often hedging occurs, the smaller the replication error.

```
/* We examine the range of possibilities, computing the replication error.
*/
```

```
// the only header you need to use QuantLib
#include <ql/quantlib.hpp>
```

```
using namespace QuantLib;
```

```
/* The ReplicationError class carries out Monte Carlo simulations to evaluate
   the outcome (the replication error) of the discrete hedging strategy over
   different, randomly generated scenarios of future stock price evolution.
```

```
*/
class ReplicationError
{
public:
    ReplicationError(Option::Type type,
                    Time maturity,
                    double strike,
                    double s0,
                    double sigma,
                    Rate r)
    : maturity_(maturity), payoff_(type, strike), s0_(s0),
      sigma_(sigma), r_(r) {

        // value of the option
        EuropeanOption option = EuropeanOption(type, s0_, strike, 0.0, r_,
                                                maturity_, sigma_);
        std::cout << "Option value: " << option.value() << std::endl;

        // store option's vega, since Derman and Kamal's formula needs it
        vega_ = option.vega();

        std::cout << std::endl;
        std::cout <<
            "      |          | P&L  \t|  P&L      | Derman&Kamal | P&L"
            "      \t| P&L" << std::endl;

        std::cout <<
```

```

        "samples | trades | Mean \t| Std Dev | Formula      |"
        " skewness \t| kurt." << std::endl;

        std::cout << "-----"
        "-----" << std::endl;
    }

    // the actual replication error computation
    void compute(int nTimeSteps, int nSamples);
private:
    Time maturity_;
    PlainVanillaPayoff payoff_;
    double s0_;
    double sigma_;
    Rate r_;
    double vega_;
};

// The key for the MonteCarlo simulation is to have a PathPricer_old that
// implements a value(const Path& path) method.
// This method prices the portfolio for each Path of the random variable
class ReplicationPathPricer : public PathPricer_old<Path>
{
public:
    // real constructor
    ReplicationPathPricer(Option::Type type,
                          double underlying,
                          double strike,
                          Rate r,
                          Time maturity,
                          double sigma)
        : PathPricer_old<Path>(1.0, false), type_(type), underlying_(underlying),
          strike_(strike), r_(r), maturity_(maturity), sigma_(sigma) {
        QL_REQUIRE(strike_ > 0.0,
                    "ReplicationPathPricer: strike must be positive");
        QL_REQUIRE(underlying_ > 0.0,
                    "ReplicationPathPricer: underlying must be positive");
        QL_REQUIRE(r_ >= 0.0,
                    "ReplicationPathPricer: risk free rate (r) must"
                    " be positive or zero");
        QL_REQUIRE(maturity_ > 0.0,
                    "ReplicationPathPricer: maturity must be positive");
        QL_REQUIRE(sigma_ >= 0.0,
                    "ReplicationPathPricer: volatility (sigma)"
                    " must be positive or zero");
    }

    // The value() method encapsulates the pricing code
    double operator()(const Path& path) const;

private:
    Option::Type type_;
    double underlying_, strike_;
    Rate r_;
    Time maturity_;
    double sigma_;
};

// Compute Replication Error as in the Derman and Kamal's research note
int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        Time maturity = 1./12.;    // 1 month
        double strike = 100;
    }
}

```

```

        double underlying = 100;
        double volatility = 0.20; // 20%
        Rate riskFreeRate = 0.05; // 5%
        ReplicationError rp(Option::Call, maturity, strike, underlying,
                           volatility, riskFreeRate);

        int scenarios = 50000;
        int hedgesNum;

        hedgesNum = 21;
        rp.compute(hedgesNum, scenarios);

        hedgesNum = 84;
        rp.compute(hedgesNum, scenarios);

        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}

/* The actual computation of the Profit&Loss for each single path.

In each scenario N reheding trades spaced evenly in time over
the life of the option are carried out, using the Black-Scholes
hedge ratio.
*/
double ReplicationPathPricer::operator()(const Path& path) const
{
    // path is an instance of QuantLib::Path
    // It contains the list of variations.
    // It can be used as an array: it has a size() method
    int n = path.size();
    QL_REQUIRE(n>0,
               "ReplicationPathPricer: the path cannot be empty");

    // discrete hedging interval
    Time dt = maturity_/n;

    // For simplicity, we assume the stock pays no dividends.
    double stockDividendYield = 0.0;

    // let's start
    Time t = 0;

    // stock value at t=0
    double stock = underlying_;
    double stockLogGrowth = 0.0;

    // money account at t=0
    double money_account = 0.0;

    /******
    *** the initial deal ***
    *****/
    // option fair price (Black-Scholes) at t=0
    EuropeanOption option = EuropeanOption(type_, stock, strike_,
                                             stockDividendYield, r_, maturity_, sigma_);
    // sell the option, cash in its premium
    money_account += option.value();
    // compute delta

```

```

double delta = option.delta();
// delta-hedge the option buying stock
double stockAmount = delta;
money_account -= stockAmount*stock;

/*****
*** hedging during option life ***
*****/
for(int step = 0; step < n-1; step++){

    // time flows
    t += dt;

    // accruing on the money account
    money_account *= QL_EXP( r_*dt );

    // stock growth:
    // path contains the list of Gaussian variations
    // and path[n] is the n-th variation
    stockLogGrowth += path[step];
    stock = underlying_*QL_EXP(stockLogGrowth);

    // recalculate option value at the current stock value,
    // and the current time to maturity
    EuropeanOption option = EuropeanOption(type_, stock, strike_,
        stockDividendYield, r_, maturity_-t, sigma_);

    // recalculate delta
    delta = option.delta();

    // re-hedging
    money_account -= (delta - stockAmount)*stock;
    stockAmount = delta;
}

/*****
*** option expiration ***
*****/
// last accrual on my money account
money_account *= QL_EXP( r_*dt );
// last stock growth
stockLogGrowth += path[n-1];
stock = underlying_*QL_EXP(stockLogGrowth);

// the hedger delivers the option payoff to the option holder
double optionPayoff = PlainVanillaPayoff(type_, strike_)(stock);
money_account -= optionPayoff;

// and unwinds the hedge selling his stock position
money_account += stockAmount*stock;

// final Profit&Loss
return money_account;
}

// The computation over nSamples paths of the P&L distribution
void ReplicationError::compute(int nTimeSteps, int nSamples)
{
    QL_REQUIRE(nTimeSteps>0,
        "ReplicationError::compute : the number of steps must be > 0");

    // hedging interval
    // double tau = maturity_ / nTimeSteps;

    /* Black-Scholes framework: the underlying stock price evolves
       lognormally with a fixed known volatility that stays constant

```

```

        throughout time.
    */
    double drift = r_ - 0.5*sigma_*sigma_;

    // Black Scholes equation rules the path generator:
    // at each step the log of the stock
    // will have drift and sigma^2 variance
    Handle<GaussianPathGenerator_old> myPathGenerator(
        new GaussianPathGenerator_old(drift, sigma_*sigma_,
            maturity_, nTimeSteps));

    // The replication strategy's Profit&Loss is computed for each path
    // of the stock. The path pricer knows how to price a path using its
    // value() method
    Handle<PathPricer_old<Path> > myPathPricer =
        (new
            ReplicationPathPricer(payoff_.optionType(), s0_,
                payoff_.strike(), r_, maturity_, sigma_));

    // a statistics accumulator for the path-dependant Profit&Loss values
    Statistics statisticsAccumulator;

    // The OneFactorMonteCarloModel generates paths using myPathGenerator
    // each path is priced using myPathPricer
    // prices will be accumulated into statisticsAccumulator
    OneFactorMonteCarloOption_old MCSimulation(myPathGenerator,
        myPathPricer, statisticsAccumulator, false);

    // the model simulates nSamples paths
    MCSimulation.addSamples(nSamples);

    // the sampleAccumulator method of OneFactorMonteCarloOption_old
    // gives access to all the methods of statisticsAccumulator
    double PLMean = MCSimulation.sampleAccumulator().mean();
    double PLStDev = MCSimulation.sampleAccumulator().standardDeviation();
    double PLSkew = MCSimulation.sampleAccumulator().skewness();
    double PLKurt = MCSimulation.sampleAccumulator().kurtosis();

    // Derman and Kamal's formula
    double theorStd = QL_SQRT(M_PI/4/nTimeSteps)*vega_*sigma_;

    std::cout << nSamples << "\t| "
        << nTimeSteps << "\t | "
        << DoubleFormatter::toString(PLMean, 3) << " \t| "
        << DoubleFormatter::toString(PLStDev, 2) << " \t | "
        << DoubleFormatter::toString(theorStd, 2) << " \t | "
        << DoubleFormatter::toString(PLSkew, 2) << " \t| "
        << DoubleFormatter::toString(PLKurt, 2) << std::endl;
}

```

11.4 EuropeanOption.cpp

This example calculates European options using different methods while testing call-put parity.

```
#include <ql/quantlib.hpp>

using namespace QuantLib;

// This will be included in the library after a bit of redesign
class WeightedPayoff {
public:
    WeightedPayoff(Option::Type type,
                  Time maturity,
                  double strike,
                  double s0,
                  double sigma,
                  Rate r,
                  Rate q)
        : type_(type), maturity_(maturity),
          strike_(strike),
          s0_(s0),
          sigma_(sigma), r_(r), q_(q){}

    double operator()(double x) const {
        double nuT = (r_-q_-0.5*sigma_*sigma_)*maturity_;
        return QL_EXP(-r_*maturity_)
            *PlainVanillaPayoff(type_, strike_)(s0_*QL_EXP(x))
            *QL_EXP(-(x - nuT)*(x - nuT)/(2*sigma_*sigma_*maturity_))
            /QL_SQRT(2.0*M_PI*sigma_*sigma_*maturity_);
    }
private:
    Option::Type type_;
    Time maturity_;
    double strike_;
    double s0_;
    double sigma_;
    Rate r_,q_;
};

int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        std::cout << "Using " << QL_VERSION << std::endl << std::endl;

        // our option
        Option::Type type(Option::Call);
        double underlying = 7;
        double strike = 8;
        Spread dividendYield = 0.05;
        Rate riskFreeRate = 0.05;

        Date todaysDate(15, May, 1998);
        Date settlementDate(17, May, 1998);
        Date exerciseDate(17, May, 1999);
        DayCounter rateDayCounter = Actual365();
        Time maturity = rateDayCounter.yearFraction(settlementDate,
            exerciseDate);

        double volatility = 0.10;
        std::cout << "option type = " << OptionTypeFormatter::toString(type)
            << std::endl;
        std::cout << "Time to maturity = " << maturity
            << std::endl;
    }
}
```

```

std::cout << "Underlying price = " << underlying
<< std::endl;
std::cout << "Strike = " << strike
<< std::endl;
std::cout << "Risk-free interest rate = " << riskFreeRate
<< std::endl;
std::cout << "dividend yield = " << dividendYield
<< std::endl;
std::cout << "Volatility = " << volatility
<< std::endl;
std::cout << std::endl;

std::string method;
double value, discrepancy, rightValue, relativeDiscrepancy;

std::cout << std::endl << std::endl ;

// write column headings
std::cout << "Method\t\tValue\tEstimatedError\tDiscrepancy"
"\tRel. Discr." << std::endl;

// first method: Black Scholes analytic solution
method = "Black Scholes";
value = EuropeanOption(type, underlying, strike,
    dividendYield, riskFreeRate, maturity, volatility).value();
double estimatedError = 0.0;
discrepancy = 0.0;
relativeDiscrepancy = 0.0;
std::cout << method << "\t"
<< DoubleFormatter::toString(value, 4) << "\t"
<< DoubleFormatter::toString(estimatedError, 4) << "\t\t"
<< DoubleFormatter::toString(discrepancy, 6) << "\t"
<< DoubleFormatter::toString(relativeDiscrepancy, 6)
<< std::endl;

// store the Black Scholes value as the correct one
rightValue = value;

// second method: Call-Put parity
method = "Call-Put parity";
Option::Type reverseType =
    (type==Option::Call ? Option::Put : Option::Call);
double coefficient =
    (type==Option::Call ? 1.0 : -1.0);
value = EuropeanOption(reverseType, underlying, strike,
    dividendYield, riskFreeRate, maturity, volatility).value()
    + coefficient * (underlying*QL_EXP(-dividendYield*maturity)
    - strike*QL_EXP(- riskFreeRate*maturity));
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
<< DoubleFormatter::toString(value, 4) << "\t"
<< "N/A\t\t"
<< discrepancy << "\t"
<< DoubleFormatter::toString(relativeDiscrepancy, 6)
<< std::endl;

// third method: Integral

```

```

method ="Integral";
WeightedPayoff po(type, maturity, strike, underlying,
                  volatility, riskFreeRate, dividendYield);
SegmentIntegral integrator(5000);

double nuT = (riskFreeRate - dividendYield
              + 0.5*volatility*volatility)*maturity;
double infinity = 10.0*volatility*QL_SQRT(maturity);

value = integrator(po, nuT-infinity, nuT+infinity);
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
          << DoubleFormatter::toString(value, 4) << "\t"
          << "N/A\t\t"
          << DoubleFormatter::toString(discrepancy, 6) << "\t"
          << DoubleFormatter::toString(relativeDiscrepancy, 6)
          << std::endl;

// fourth method: Finite Differences
method ="Finite Diff.";
Size grid = 100;
value = FdEuropean(type, underlying, strike,
                  dividendYield, riskFreeRate, maturity, volatility, grid).value();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
          << DoubleFormatter::toString(value, 4) << "\t"
          << "N/A\t\t"
          << DoubleFormatter::toString(discrepancy, 6) << "\t"
          << DoubleFormatter::toString(relativeDiscrepancy, 6)
          << std::endl;

/*****

// New option pricing framework
std::cout << "\nNew Pricing engine framework" << std::endl;

Date midlifeDate(19, November, 1998);
std::vector<Date> exDates(2);
exDates[0]=midlifeDate;
exDates[1]=exerciseDate;

Handle<Exercise> exercise(new EuropeanExercise(exerciseDate));
Handle<Exercise> amExercise(new AmericanExercise(settlementDate,
                                                exerciseDate));
Handle<Exercise> berExercise(new BermudanExercise(exDates));

RelinkableHandle<Quote> underlyingH(
    Handle<Quote>(new SimpleQuote(underlying)));

// bootstrap the yield/dividend/vol curves
RelinkableHandle<TermStructure> flatTermStructure(
    Handle<TermStructure>(
        new FlatForward(todaysDate, settlementDate,
                        riskFreeRate, rateDayCounter)));
RelinkableHandle<TermStructure> flatDividendTS(
    Handle<TermStructure>(
        new FlatForward(todaysDate, settlementDate,
                        dividendYield, rateDayCounter)));
RelinkableHandle<BlackVolTermStructure> flatVolTS(
    Handle<BlackVolTermStructure>(

```

```

        new BlackConstantVol(settlementDate, volatility)));

    std::vector<Date> dates(4);
    dates[0] = settlementDate.plusMonths(1);
    dates[1] = exerciseDate;
    dates[2] = exerciseDate.plusMonths(6);
    dates[3] = exerciseDate.plusMonths(12);
    std::vector<double> strikes(4);
    strikes[0] = underlying*0.9;
    strikes[1] = underlying;
    strikes[2] = underlying*1.1;
    strikes[3] = underlying*1.2;

    Matrix vols(4,4);
    vols[0][0] = volatility*1.1;
        vols[0][1] = volatility;
            vols[0][2] = volatility*0.9;
                vols[0][3] = volatility*0.8;
    vols[1][0] = volatility*1.1;
        vols[1][1] = volatility;
            vols[1][2] = volatility*0.9;
                vols[1][3] = volatility*0.8;
    vols[2][0] = volatility*1.1;
        vols[2][1] = volatility;
            vols[2][2] = volatility*0.9;
                vols[2][3] = volatility*0.8;
    vols[3][0] = volatility*1.1;
        vols[3][1] = volatility;
            vols[3][2] = volatility*0.9;
                vols[3][3] = volatility*0.8;

    RelinkableHandle<BlackVolTermStructure> blackSurface(
        Handle<BlackVolTermStructure>(
            new BlackVarianceSurface(settlementDate, dates,
                                    strikes, vols)));

    Handle<StrikedTypePayoff> payoff(new
        PlainVanillaPayoff(type, strike));

    Handle<BlackScholesStochasticProcess> stochasticProcess(new
        BlackScholesStochasticProcess(underlyingH, flatDividendTS,
        flatTermStructure,
        // blackSurface
        flatVolTS));

    VanillaOption option(stochasticProcess, payoff, exercise,
        Handle<PricingEngine>(new AnalyticEuropeanEngine()));

    // method: Black Scholes Engine
    method = "Black Scholes";
    option.setPricingEngine(Handle<PricingEngine>(
        new AnalyticEuropeanEngine()));
    value = option.NPV();
    discrepancy = QL_FABS(value-rightValue);
    relativeDiscrepancy = discrepancy/rightValue;
    std::cout << method << "\t"
        << DoubleFormatter::toString(value, 4) << "\t"
        << "N/A\t\t"
        << DoubleFormatter::toString(discrepancy, 6) << "\t"
        << DoubleFormatter::toString(relativeDiscrepancy, 6)
        << std::endl;

    // method: Integral
    method = "Integral";

```

```

option.setPricingEngine(Handle<PricingEngine>(
    new IntegralEngine()));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DoubleFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

/*

// method: Integral
method = "Binary Cash";
option.setPricingEngine(Handle<PricingEngine>(
    new IntegralCashOrNothingEngine(1.0)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DoubleFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// method: Integral
method = "Binary Asset";
option.setPricingEngine(Handle<PricingEngine>(
    new IntegralAssetOrNothingEngine()));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DoubleFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

*/

Size timeSteps = 801;

// Binomial Method (JR)
method = "Binomial (JR)";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DoubleFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Binomial Method (CRR)
method = "Binomial (CRR)";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"

```

```

        << DoubleFormatter::toString(value, 4) << "\t"
        << "N/A\t\t"
        << DoubleFormatter::toString(discrepancy, 6) << "\t"
        << DoubleFormatter::toString(relativeDiscrepancy, 6)
        << std::endl;

// Equal Probability Additive Binomial Tree (EQP)
method = "Additive (EQP)";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<AdditiveEQPBinoimialTree>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
        << DoubleFormatter::toString(value, 4) << "\t"
        << "N/A\t\t"
        << DoubleFormatter::toString(discrepancy, 6) << "\t"
        << DoubleFormatter::toString(relativeDiscrepancy, 6)
        << std::endl;

// Equal Jumps Additive Binomial Tree (Trigeorgis)
method = "Bin. Trigeorgis";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
        << DoubleFormatter::toString(value, 4) << "\t"
        << "N/A\t\t"
        << DoubleFormatter::toString(discrepancy, 6) << "\t"
        << DoubleFormatter::toString(relativeDiscrepancy, 6)
        << std::endl;

// Tian Binomial Tree (third moment matching)
method = "Binomial Tian";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<Tian>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
        << DoubleFormatter::toString(value, 4) << "\t"
        << "N/A\t\t"
        << DoubleFormatter::toString(discrepancy, 6) << "\t"
        << DoubleFormatter::toString(relativeDiscrepancy, 6)
        << std::endl;

// Leisen-Reimer Binomial Tree
method = "Binomial LR";
option.setPricingEngine(Handle<PricingEngine>(
    new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
        << DoubleFormatter::toString(value, 4) << "\t"
        << "N/A\t\t"
        << DoubleFormatter::toString(discrepancy, 6) << "\t"
        << DoubleFormatter::toString(relativeDiscrepancy, 6)
        << std::endl;

// Finite Differences Method: not implemented

/*method = "Finite Diff.";
option.setPricingEngine(Handle<PricingEngine>(
    new FdVanillaEngine()));

```

```

value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DoubleFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;*/

// Monte Carlo Method
timeSteps = 1;

method = "MC (crude)";
Size mcSeed = 42;

#if defined(QL_PATCH_MICROSOFT)
Handle<PricingEngine> mcengine1(
    new MCEuropeanEngine<PseudoRandom>(timeSteps, false, false,
                                         Null<int>(), 0.02,
                                         Null<int>(), mcSeed));
#else
Handle<PricingEngine> mcengine1 =
    MakeMCEuropeanEngine<PseudoRandom>().withStepsPerYear(timeSteps)
                                         .withTolerance(0.02)
                                         .withSeed(mcSeed);
#endif
option.setPricingEngine(mcengine1);

value = option.NPV();
double errorEstimate = option.errorEstimate();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DoubleFormatter::toString(value, 4) << "\t"
    << DoubleFormatter::toString(errorEstimate, 4) << "\t\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

method = "MC (Sobol)";
timeSteps = 1;
Size nSamples = 32768; // 2^15

#if defined(QL_PATCH_MICROSOFT)
Handle<PricingEngine> mcengine2(
    new MCEuropeanEngine<LowDiscrepancy>(timeSteps, false, false,
                                         nSamples, Null<double>(),
                                         Null<int>()));
#else
Handle<PricingEngine> mcengine2 =
    MakeMCEuropeanEngine<LowDiscrepancy>().withStepsPerYear(timeSteps)
                                         .withSamples(nSamples);
#endif
option.setPricingEngine(mcengine2);

value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DoubleFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DoubleFormatter::toString(discrepancy, 6) << "\t"
    << DoubleFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

```

```
        return 0;
    } catch (std::exception& e) {
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```

11.5 history_iterators.cpp

This code exemplifies how to use History iterators to perform gaussianstatistics analyses on historical data.

```
// initialize a History
History h(...);

// print out the mean value and its standard deviation.

GaussianStatistics s;
s.addSequence(h.vdbegin(),h.vdend());
cout << "Historical mean: " << s.mean() << endl;
cout << "Std. deviation: " << s.standardDeviation() << endl;

// Another possibility: print out the maximum value.

History::const_valid_iterator max = h.vbegin(), i=max, end = h.vend();
for (i++; i!=end; i++)
    if (i->value() > max->value())
        max = i;
cout << "Maximum value: " << max->value()
    << " assumed " << DateFormatter::toString(max->date()) << endl;

// or the minimum, this time the STL way:

bool lessthan(const History::Entry& i, const History::Entry& j) {
    return i.value() < j.value();
}

History::const_valid_iterator min =
    std::min_element(h.vbegin(),h.vend(),lessthan);
cout << "Minimum value: " << min->value()
    << " assumed " << DateFormatter::toString(min->date()) << endl;
```

11.6 swapvaluation.cpp

This is an example of using the QuantLib Term Structure for pricing a simple swap.

```

/* This example shows how to set up a Term Structure and then price a simple
   swap.
*/

// the only header you need to use QuantLib
#include <ql/quantlib.hpp>
#include <iomanip>

using namespace QuantLib;

int main(int argc, char* argv[])
{
    try {
        QL_IO_INIT

        /*****
         *** MARKET DATA ***
         *****/

        Calendar calendar = TARGET();
        Date todaysDate(6, November, 2001);
        Date settlementDate(8, November, 2001);

        // deposits
        double d1wQuote=0.0382;
        double d1mQuote=0.0372;
        double d3mQuote=0.0363;
        double d6mQuote=0.0353;
        double d9mQuote=0.0348;
        double d1yQuote=0.0345;
        // FRAs
        double fra3x6Quote=0.037125;
        double fra6x9Quote=0.037125;
        double fra6x12Quote=0.037125;
        // futures
        double fut1Quote=96.2875;
        double fut2Quote=96.7875;
        double fut3Quote=96.9875;
        double fut4Quote=96.6875;
        double fut5Quote=96.4875;
        double fut6Quote=96.3875;
        double fut7Quote=96.2875;
        double fut8Quote=96.0875;
        // swaps
        double s2yQuote=0.037125;
        double s3yQuote=0.0398;
        double s5yQuote=0.0443;
        double s10yQuote=0.05165;
        double s15yQuote=0.055175;

        /*****
         *** QUOTES ***
         *****/

        // SimpleQuote stores a value which can be manually changed;
        // other Quote subclasses could read the value from a database
        // or some kind of data feed.

        // deposits
        Handle<Quote> d1wRate(new SimpleQuote(d1wQuote));
        Handle<Quote> d1mRate(new SimpleQuote(d1mQuote));

```

```

Handle<Quote> d3mRate(new SimpleQuote(d3mQuote));
Handle<Quote> d6mRate(new SimpleQuote(d6mQuote));
Handle<Quote> d9mRate(new SimpleQuote(d9mQuote));
Handle<Quote> d1yRate(new SimpleQuote(d1yQuote));
// FRAs
Handle<Quote> fra3x6Rate(new SimpleQuote(fra3x6Quote));
Handle<Quote> fra6x9Rate(new SimpleQuote(fra6x9Quote));
Handle<Quote> fra6x12Rate(new SimpleQuote(fra6x12Quote));
// futures
Handle<Quote> fut1Price(new SimpleQuote(fut1Quote));
Handle<Quote> fut2Price(new SimpleQuote(fut2Quote));
Handle<Quote> fut3Price(new SimpleQuote(fut3Quote));
Handle<Quote> fut4Price(new SimpleQuote(fut4Quote));
Handle<Quote> fut5Price(new SimpleQuote(fut5Quote));
Handle<Quote> fut6Price(new SimpleQuote(fut6Quote));
Handle<Quote> fut7Price(new SimpleQuote(fut7Quote));
Handle<Quote> fut8Price(new SimpleQuote(fut8Quote));
// swaps
Handle<Quote> s2yRate(new SimpleQuote(s2yQuote));
Handle<Quote> s3yRate(new SimpleQuote(s3yQuote));
Handle<Quote> s5yRate(new SimpleQuote(s5yQuote));
Handle<Quote> s10yRate(new SimpleQuote(s10yQuote));
Handle<Quote> s15yRate(new SimpleQuote(s15yQuote));

/*****
***   RATE HELPERS   ***
*****/

// RateHelpers are built from the above quotes together with
// other instrument dependant infos. Quotes are passed in
// relinkable handles which could be relinked to some other
// data source later.

// deposits
DayCounter depositDayCounter = Actual360();
int settlementDays = 2;

Handle<RateHelper> d1w(new DepositRateHelper(
    RelinkableHandle<Quote>(d1wRate),
    1, Weeks, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d1m(new DepositRateHelper(
    RelinkableHandle<Quote>(d1mRate),
    1, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d3m(new DepositRateHelper(
    RelinkableHandle<Quote>(d3mRate),
    3, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d6m(new DepositRateHelper(
    RelinkableHandle<Quote>(d6mRate),
    6, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d9m(new DepositRateHelper(
    RelinkableHandle<Quote>(d9mRate),
    9, Months, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));
Handle<RateHelper> d1y(new DepositRateHelper(
    RelinkableHandle<Quote>(d1yRate),
    1, Years, settlementDays,
    calendar, ModifiedFollowing, depositDayCounter));

// setup FRAs
Handle<RateHelper> fra3x6(new FraRateHelper(
    RelinkableHandle<Quote>(fra3x6Rate),

```

```

        3, 6, settlementDays, calendar, ModifiedFollowing,
        depositDayCounter));
Handle<RateHelper> fra6x9(new FraRateHelper(
    RelinkableHandle<Quote>(fra6x9Rate),
    6, 9, settlementDays, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fra6x12(new FraRateHelper(
    RelinkableHandle<Quote>(fra6x12Rate),
    6, 12, settlementDays, calendar, ModifiedFollowing,
    depositDayCounter));

// setup futures
int futMonths = 3;
Handle<RateHelper> fut1(new FuturesRateHelper(
    RelinkableHandle<Quote>(fut1Price),
    Date(19, December, 2001),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut2(new FuturesRateHelper(
    RelinkableHandle<Quote>(fut1Price),
    Date(20, March, 2002),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut3(new FuturesRateHelper(
    RelinkableHandle<Quote>(fut1Price),
    Date(19, June, 2002),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut4(new FuturesRateHelper(
    RelinkableHandle<Quote>(fut1Price),
    Date(18, September, 2002),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut5(new FuturesRateHelper(
    RelinkableHandle<Quote>(fut1Price),
    Date(18, December, 2002),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut6(new FuturesRateHelper(
    RelinkableHandle<Quote>(fut1Price),
    Date(19, March, 2003),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut7(new FuturesRateHelper(
    RelinkableHandle<Quote>(fut1Price),
    Date(18, June, 2003),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
Handle<RateHelper> fut8(new FuturesRateHelper(
    RelinkableHandle<Quote>(fut1Price),
    Date(17, September, 2003),
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));

// setup swaps
int swFixedLegFrequency = 1;
bool swFixedLegIsAdjusted = false;
DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
int swFloatingLegFrequency = 2;

Handle<RateHelper> s2y(new SwapRateHelper(
    RelinkableHandle<Quote>(s2yRate),
    2, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,

```

```

        swFloatingLegFrequency));
Handle<RateHelper> s3y(new SwapRateHelper(
    RelinkableHandle<Quote>(s3yRate),
    3, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
Handle<RateHelper> s5y(new SwapRateHelper(
    RelinkableHandle<Quote>(s5yRate),
    5, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
Handle<RateHelper> s10y(new SwapRateHelper(
    RelinkableHandle<Quote>(s10yRate),
    10, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));
Handle<RateHelper> s15y(new SwapRateHelper(
    RelinkableHandle<Quote>(s15yRate),
    15, Years, settlementDays,
    calendar, ModifiedFollowing, swFixedLegFrequency,
    swFixedLegIsAdjusted, swFixedLegDayCounter,
    swFloatingLegFrequency));

/*****
** CURVE BUILDING **
*****/

// Any DayCounter would be fine.
// ActualActual::ISDA ensures that 30 years is 30.0
DayCounter termStructureDayCounter =
    ActualActual(ActualActual::ISDA);

// A depo-swap curve
std::vector<Handle<RateHelper> > depoSwapInstruments;
depoSwapInstruments.push_back(d1w);
depoSwapInstruments.push_back(d1m);
depoSwapInstruments.push_back(d3m);
depoSwapInstruments.push_back(d6m);
depoSwapInstruments.push_back(d9m);
depoSwapInstruments.push_back(d1y);
depoSwapInstruments.push_back(s2y);
depoSwapInstruments.push_back(s3y);
depoSwapInstruments.push_back(s5y);
depoSwapInstruments.push_back(s10y);
depoSwapInstruments.push_back(s15y);
Handle<TermStructure> depoSwapTermStructure(new
    PiecewiseFlatForward(todaysDate, settlementDate,
        depoSwapInstruments, termStructureDayCounter));

// A depo-futures-swap curve
std::vector<Handle<RateHelper> > depoFutSwapInstruments;
depoFutSwapInstruments.push_back(d1w);
depoFutSwapInstruments.push_back(d1m);
depoFutSwapInstruments.push_back(fut1);
depoFutSwapInstruments.push_back(fut2);
depoFutSwapInstruments.push_back(fut3);
depoFutSwapInstruments.push_back(fut4);
depoFutSwapInstruments.push_back(fut5);
depoFutSwapInstruments.push_back(fut6);
depoFutSwapInstruments.push_back(fut7);
depoFutSwapInstruments.push_back(fut8);

```

```

depoFutSwapInstruments.push_back(s3y);
depoFutSwapInstruments.push_back(s5y);
depoFutSwapInstruments.push_back(s10y);
depoFutSwapInstruments.push_back(s15y);
Handle<TermStructure> depoFutSwapTermStructure(new
    PiecewiseFlatForward(todaysDate, settlementDate,
        depoFutSwapInstruments, termStructureDayCounter));

// A depo-FRA-swap curve
std::vector<Handle<RateHelper> > depoFRASwapInstruments;
depoFRASwapInstruments.push_back(d1w);
depoFRASwapInstruments.push_back(d1m);
depoFRASwapInstruments.push_back(d3m);
depoFRASwapInstruments.push_back(fra3x6);
depoFRASwapInstruments.push_back(fra6x9);
depoFRASwapInstruments.push_back(fra6x12);
depoFRASwapInstruments.push_back(s2y);
depoFRASwapInstruments.push_back(s3y);
depoFRASwapInstruments.push_back(s5y);
depoFRASwapInstruments.push_back(s10y);
depoFRASwapInstruments.push_back(s15y);
Handle<TermStructure> depoFRASwapTermStructure(new
    PiecewiseFlatForward(todaysDate, settlementDate,
        depoFRASwapInstruments, termStructureDayCounter));

// Term structures that will be used for pricing:
// the one used for discounting cash flows
RelinkableHandle<TermStructure> discountingTermStructure;
// the one used for forward rate forecasting
RelinkableHandle<TermStructure> forecastingTermStructure;

/*****
 * SWAPS TO BE PRICED *
*****/

// constant nominal 1,000,000 Euro
double nominal = 1000000.0;
// fixed leg
int fixedLegFrequency = 1; // annual
bool fixedLegIsAdjusted = false;
RollingConvention roll = ModifiedFollowing;
DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
int fixingDays = 2;
Rate fixedRate = 0.04;

// floating leg
int floatingLegFrequency = 2;
Handle<Xibor> euriborIndex(new Euribor(6, Months,
    forecastingTermStructure)); // using the forecasting curve
Spread spread = 0.0;

int lenghtInYears = 5;
bool payFixedRate = true;
SimpleSwap spot5YearSwap(payFixedRate, settlementDate, lenghtInYears,
    Years, calendar, roll, nominal, fixedLegFrequency, fixedRate,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    euriborIndex, fixingDays, spread,
    discountingTermStructure); // using the discounting curve
SimpleSwap oneYearForward5YearSwap(payFixedRate,
    calendar.advance(settlementDate, 1, Years, ModifiedFollowing),
    lenghtInYears, Years,
    calendar, roll, nominal, fixedLegFrequency, fixedRate,
    fixedLegIsAdjusted, fixedLegDayCounter, floatingLegFrequency,
    euriborIndex, fixingDays, spread,

```

```

        discountingTermStructure); // using the discounting curve

/*****
 * SWAP PRICING *
*****/

// utilities for reporting
std::vector<std::string> headers(4);
headers[0] = "term structure";
headers[1] = "net present value";
headers[2] = "fair spread";
headers[3] = "fair fixed rate";
std::string separator = " | ";
Size width = headers[0].size() + separator.size()
             + headers[1].size() + separator.size()
             + headers[2].size() + separator.size()
             + headers[3].size() + separator.size() - 1;
std::string rule(width, '-'), dblrule(width, '=');
std::string tab(8, ' ');

// calculations

std::cout << dblrule << std::endl;
std::cout << "5-year market swap-rate = "
          << RateFormatter::toString(s5yRate->value(),2) << std::endl;
std::cout << dblrule << std::endl;

std::cout << tab << "5-years swap paying "
          << RateFormatter::toString(fixedRate,2) << std::endl;
std::cout << headers[0] << separator
          << headers[1] << separator
          << headers[2] << separator
          << headers[3] << separator << std::endl;
std::cout << rule << std::endl;

double NPV;
Rate fairRate;
Spread fairSpread;

// Of course, you're not forced to really use different curves
forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-swap" << separator;
std::cout << std::setw(headers[1].size())
          << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
          << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
          << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

// let's check that the 5 years swap has been correctly re-priced
QL_REQUIRE(QL_FABS(fairRate-s5yQuote)<1e-8,
            "5-years swap mispriced!");

forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);

```

```

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
  << "depo-fut-swap" << separator;
std::cout << std::setw(headers[1].size())
  << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
  << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
  << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

QL_REQUIRE(QL_FABS(fairRate-s5yQuote)<1e-8,
  "5-years swap mispriced!");

forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
  << "depo-FRA-swap" << separator;
std::cout << std::setw(headers[1].size())
  << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
  << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
  << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

QL_REQUIRE(QL_FABS(fairRate-s5yQuote)<1e-8,
  "5-years swap mispriced!");

std::cout << rule << std::endl;

// now let's price the 1Y forward 5Y swap

std::cout << tab << "5-years, 1-year forward swap paying "
  << RateFormatter::toString(fixedRate,2) << std::endl;
std::cout << headers[0] << separator
  << headers[1] << separator
  << headers[2] << separator
  << headers[3] << separator << std::endl;
std::cout << rule << std::endl;

forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
  << "depo-swap" << separator;
std::cout << std::setw(headers[1].size())
  << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
  << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
  << RateFormatter::toString(fairRate,4) << separator;

```

```

std::cout << std::endl;

forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-fut-swap" << separator;
std::cout << std::setw(headers[1].size())
          << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
          << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
          << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
          << "depo-FRA-swap" << separator;
std::cout << std::setw(headers[1].size())
          << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
          << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
          << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

// now let's say that the 5-years swap rate goes up to 4.60%.
// A smarter market element--say, connected to a data source-- would
// notice the change itself. Since we're using SimpleQuotes,
// we'll have to change the value manually--which forces us to
// downcast the handle and use the SimpleQuote
// interface. In any case, the point here is that a change in the
// value contained in the Quote triggers a new bootstrapping
// of the curve and a repricing of the swap.

#ifdef HAVE_BOOST
Handle<SimpleQuote> fiveYearsRate =
    boost::dynamic_pointer_cast<SimpleQuote>(s5yRate);
#else
Handle<SimpleQuote> fiveYearsRate = s5yRate;
#endif
fiveYearsRate->setValue(0.0460);

std::cout << dblrule << std::endl;
std::cout << "5-year market swap-rate = "
          << RateFormatter::toString(s5yRate->value(),2) << std::endl;
std::cout << dblrule << std::endl;

std::cout << tab << "5-years swap paying "
          << RateFormatter::toString(fixedRate,2) << std::endl;
std::cout << headers[0] << separator
          << headers[1] << separator
          << headers[2] << separator
          << headers[3] << separator << std::endl;

```

```

std::cout << rule << std::endl;

// now get the updated results
forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
    << "depo-swap" << separator;
std::cout << std::setw(headers[1].size())
    << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

QL_REQUIRE(QL_FABS(fairRate-s5yRate->value())<1e-8,
    "5-years swap mispriced!");

forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
    << "depo-fut-swap" << separator;
std::cout << std::setw(headers[1].size())
    << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

QL_REQUIRE(QL_FABS(fairRate-s5yRate->value())<1e-8,
    "5-years swap mispriced!");

forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);

NPV = spot5YearSwap.NPV();
fairSpread = spot5YearSwap.fairSpread();
fairRate = spot5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
    << "depo-FRA-swap" << separator;
std::cout << std::setw(headers[1].size())
    << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

QL_REQUIRE(QL_FABS(fairRate-s5yRate->value())<1e-8,
    "5-years swap mispriced!");

std::cout << rule << std::endl;

```

```

// the 1Y forward 5Y swap changes as well

std::cout << tab << "5-years, 1-year forward swap paying "
    << RateFormatter::toString(fixedRate,2) << std::endl;
std::cout << headers[0] << separator
    << headers[1] << separator
    << headers[2] << separator
    << headers[3] << separator << std::endl;
std::cout << rule << std::endl;

forecastingTermStructure.linkTo(depoSwapTermStructure);
discountingTermStructure.linkTo(depoSwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
    << "depo-swap" << separator;
std::cout << std::setw(headers[1].size())
    << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

forecastingTermStructure.linkTo(depoFutSwapTermStructure);
discountingTermStructure.linkTo(depoFutSwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
    << "depo-fut-swap" << separator;
std::cout << std::setw(headers[1].size())
    << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

forecastingTermStructure.linkTo(depoFRASwapTermStructure);
discountingTermStructure.linkTo(depoFRASwapTermStructure);

NPV = oneYearForward5YearSwap.NPV();
fairSpread = oneYearForward5YearSwap.fairSpread();
fairRate = oneYearForward5YearSwap.fairRate();

std::cout << std::setw(headers[0].size())
    << "depo-FRA-swap" << separator;
std::cout << std::setw(headers[1].size())
    << DoubleFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread,4) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate,4) << separator;
std::cout << std::endl;

return 0;

} catch (std::exception& e) {

```

```
        std::cout << e.what() << std::endl;
        return 1;
    } catch (...) {
        std::cout << "unknown error" << std::endl;
        return 1;
    }
}
```


Chapter 12

Todo List

Class AmericanCondition(p. 115) Unify the intrinsicValues/Payoff thing

Class AmericanExercise(p. 116) check that everywhere the American condition is applied from the earliestDate and not earlier

Class AmericanPayoffAtExpiry(p. 117) calculate greeks

Class AmericanPayoffAtHit(p. 118) calculate greeks

Class AnalyticDigitalAmericanEngine(p. 121) add more greeks (as of now only delta and rho available)

Class AUDLibor(p. 131) check settlement days

Class BermudanExercise(p. 143) it would be nice to have a way for making a Bermudan with one exercise date equivalent to an European

Class BicubicSpline(p. 144) revise end conditions

Class BivariateCumulativeNormalDistribution(p. 152) check accuracy of this algorithm and compare with: 1) Drezner, Z, (1978), Computation of the bivariate normal integral, Mathematics of Computation 32, pp. 277-279. 2) Drezner, Z. and Wesolowsky, G. O. (1990) 'On the Computation of the Bivariate Normal Integral', Journal of Statistical Computation and Simulation 35, pp. 101-107. 3) Drezner, Z (1992) Computation of the Multivariate Normal Integral, ACM Transactions on Mathematics Software 18, pp. 450-460. 4) Drezner, Z (1994) Computation of the Trivariate Normal Integral, Mathematics of Computation 62, pp. 289-294. 5) Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.

Class BlackScholesProcess(p. 162) revise extrapolation

Class BlackVarianceCurve(p. 164) check time extrapolation

Class BlackVarianceSurface(p. 166) check time extrapolation

Member Side(p. 172) Generalize for n-dimensional conditions

Class CADLibor(p. 183) check settlement days

Class CapFlatVolatilityVector(p. 194) Either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the length vector but an interpolation pointing to the original ones.

Class CHFLibor(p. 202) check settlement days and day-count

Class ContinuousGeometricAPO(p. 217) add Average Strike version and make it backward starting

Class DirichletBC(p. 248) generalize to time-dependent conditions.

Class DiscreteGeometricAPO(p. 258) add analytical greeks

Class DiscreteGeometricASO(p. 259) add analytical greeks

Class EarlyExercise(p. 272) derive a plain American Exercise class (no earliestDate, no payoff-AtExpiry)

Class ExplicitEuler(p. 284) add Richardson extrapolation

Class FraRateHelper(p. 318) convexity adjustment should be implemented.

Class GenericRiskStatistics(p. 334) add historical annualized volatility

Class IntegralEngine(p. 371) define tolerance for calculate()

Class JPYLibor(p. 383) check settlement days

Class LogLinearInterpolation(p. 414) Implement primitive, derivative, and secondDerivative functions.

Member pseudoSqrt(p. 420)(const Matrix &, SalvagingAlgorithm::Type) a) 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" b) implement Higham algorithm: Higham "Computing the nearest correlation matrix"

Class McDiscreteArithmeticAPO(p. 430) Continuous-averaging version

Class McDiscreteArithmeticASO(p. 431) Continuous Averaging version

Class MixedScheme(p. 445) a) derive variable theta schemes b) introduce multi time-level schemes.

Class MultiPath(p. 455) a) make it time-aware b) rename it as MultiAssetPath

Class MultiPathGenerator(p. 456) why store correlation Matrix rather than covariance?

Class NeumannBC(p. 460) generalize to time-dependent conditions.

Class Option::arguments(p. 491) a) remove std::vector<Time> stoppingTimes b) how to handle strike-less option (asian average strike, forward, etc.)?

Class Path(p. 501) should Path include the t=0.0 point? Alternatively all path pricers must be revisited.

Class ShoutCondition(p. 552) Unify the intrinsicValues/Payoff thing

Class Solver1D(p. 565) a) Clean up the interface so that it is clear whether the accuracy is specified for x or f(x). b) Add target value (now the target value is 0.0)

Class SwaptionVolatilityMatrix(p. 589) Either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the exercise date and length vector but an interpolation pointing to the original ones.

Class TermStructure(p. 594) add derived class ParSwapTermStructure similar to ZeroYieldTermStructure, DiscountStructure, ForwardRateStructure
allow for different compounding rules and compounding frequencies

Class TimeGrid(p. 601) What was the rationale for limiting the grid to positive times? Investigate and see whether we can use it for negative ones as well.

Class ZARLibor(p. 634) check settlement days

Chapter 13

Deprecated List

Member BoxMullerGaussianRng(p. [175](#))(**long seed=0**) initialize with a random number generator instead.

Member CLGaussianRng(p. [203](#))(**long seed=0**) initialize with a random number generator instead.

Class EuropeanOption(p. [282](#)) use VanillaOption with EuropeanAnalyticEngine

Member ICGaussianRng(p. [350](#))(**long seed=0**) initialize with a random number generator instead.

Member matrixSqrt(p. [421](#))(**const Matrix &**) use CholeskyDecomposition or pseudoSqrt instead

Class PathGenerator_old(p. [503](#)) use PathGenerator instead

Class PathPricer_old(p. [505](#)) use PathPricer instead

Class RandomArrayGenerator(p. [533](#)) use RandomSequenceGenerator instead.

Chapter 14

Bug List

Class BPSBasketCalculator(p. 176) This class must still be checked. It is not guaranteed to yield the right results.

Class CoxIngersollRoss(p. 224) This class was not tested enough to guarantee its functionality.

Class ExtendedCoxIngersollRoss(p. 285) This class was not tested enough to guarantee its functionality.

Class FdDividendAmericanOption(p. 296) sometimes yields negative vega when deeply in-the-money
method `impliedVolatility()` utterly fails

Class G2(p. 321) This class was not tested enough to guarantee its functionality.

Class LocalVolSurface(p. 410) This class is untested, probably unreliable.

Class MCAmericanBasketEngine(p. 422) This engine does not yet work for put options. More problems might surface.

Member impliedVolatility(p. 477)(double price, double accuracy=1.0e-4, Size maxEvaluations=100, double minV) run-time crashes are possible with the Borland compiler

Member sensitivity(p. 582)(int basis=2) const This method must still be checked. It is not guaranteed to yield the right results.

Index

- add
 - QuantLib::GeneralStatistics, 331
 - QuantLib::IncrementalStatistics, 362
- addSequence
 - QuantLib::IncrementalStatistics, 362
- advance
 - QuantLib::Calendar, 186
- amount
 - QuantLib::CashFlow, 200
 - QuantLib::FixedRateCoupon, 304
 - QuantLib::IndexedCoupon, 365
 - QuantLib::ParCoupon, 500
 - QuantLib::Short, 548
 - QuantLib::SimpleCashFlow, 553
- applyAfterApplying
 - QuantLib::BoundaryCondition, 172
- applyAfterSolving
 - QuantLib::BoundaryCondition, 172
- applyBeforeApplying
 - QuantLib::BoundaryCondition, 172
- applyBeforeSolving
 - QuantLib::BoundaryCondition, 172
- averageShortfall
 - QuantLib::GenericRiskStatistics, 336
- BicubicSpline
 - QuantLib::BicubicSpline, 144
- BilinearInterpolation
 - QuantLib::BilinearInterpolation, 146
- blackVarianceImpl
 - QuantLib::BlackVolatilityTerm-Structure, 169
- blackVollImpl
 - QuantLib::BlackVarianceTerm-Structure, 168
- BoundaryCondition
 - QuantLib::CubicSpline, 231
- BoxMullerGaussianRng
 - QuantLib::BoxMullerGaussianRng, 175
- calculate
 - QuantLib::Instrument, 368
 - QuantLib::LazyObject, 392
- Calendar
 - QuantLib::Calendar, 185
- calibrate
 - QuantLib::ShortRateModel, 551
- Character functions, 101
- CLGaussianRng
 - QuantLib::CLGaussianRng, 203
- compoundForwardImpl
 - QuantLib::DiscountStructure, 251
 - QuantLib::ExtendedDiscountCurve, 289
 - QuantLib::ForwardRateStructure, 311
 - QuantLib::ZeroYieldStructure, 638
- Coupon
 - QuantLib::Coupon, 223
- CubicSpline
 - QuantLib::CubicSpline, 231
- DayCounter
 - QuantLib::DayCounter, 241
- DEFINE_SEQUENCE_STAT_CONST_-METHOD_DOUBLE
 - sequencestatistics.hpp, 788
- DEFINE_SEQUENCE_STAT_CONST_-METHOD_VOID
 - sequencestatistics.hpp, 788
- diffusion
 - QuantLib::BlackScholesProcess, 162
 - QuantLib::DiffusionProcess, 246
 - QuantLib::OrnsteinUhlenbeckProcess, 493
 - QuantLib::SquareRootProcess, 567
- discountImpl
 - QuantLib::ForwardRateStructure, 311
 - QuantLib::ZeroYieldStructure, 638
- downsideDeviation
 - QuantLib::GenericRiskStatistics, 335
 - QuantLib::IncrementalStatistics, 361
- downsideVariance
 - QuantLib::GenericRiskStatistics, 334
 - QuantLib::IncrementalStatistics, 361
- Error
 - QuantLib::Error, 277
- errorEstimate
 - QuantLib::GeneralStatistics, 330
 - QuantLib::IncrementalStatistics, 361

- errors.hpp
 - QL_ASSERT, 691
 - QL_ENSURE, 692
 - QL_FAIL, 691
 - QL_REQUIRE, 692
- expectation
 - QuantLib::DiffusionProcess, 246
 - QuantLib::OrnsteinUhlenbeckProcess, 493
- expectationValue
 - QuantLib::GeneralStatistics, 331
- expectedShortfall
 - QuantLib::GenericRiskStatistics, 335
- FirstDerivative
 - QuantLib::CubicSpline, 231
- fixing
 - QuantLib::Index, 363
 - QuantLib::Xibor, 632
- formula
 - QuantLib::BlackModel, 159
- forwardImpl
 - QuantLib::DiscountStructure, 251
 - QuantLib::ZeroSpreadedTerm-Structure, 637
 - QuantLib::ZeroYieldStructure, 638
- freeze
 - QuantLib::LazyObject, 392
- gaussianDownsideDeviation
 - QuantLib::GaussianStatistics, 326
- gaussianDownsideVariance
 - QuantLib::GaussianStatistics, 326
- gaussianExpectedShortfall
 - QuantLib::GaussianStatistics, 327
- gaussianPercentile
 - QuantLib::GaussianStatistics, 327
- gaussianPotentialUpside
 - QuantLib::GaussianStatistics, 327
- gaussianRegret
 - QuantLib::GaussianStatistics, 327
- gaussianValueAtRisk
 - QuantLib::GaussianStatistics, 327
- Global QuantLib macros, 95
- Handle
 - QuantLib::Handle, 340
- History
 - QuantLib::History, 343
- ICGaussianRng
 - QuantLib::ICGaussianRng, 350
- impliedVolatility
 - QuantLib::OneAssetOption, 477
 - QuantLib::SingleAssetOption, 563
- Input/output functions, 102
- isBusinessDay
 - QuantLib::Calendar, 185
- isEndOfMonth
 - QuantLib::Calendar, 185
- isHoliday
 - QuantLib::Calendar, 186
- Iterator support, 106
- iteratorMacros
 - QL_FULL_ITERATOR_SUPPORT, 106
 - QL_ITERATOR, 106
 - QL_ITERATOR_TRAITS, 106
 - QL_REVERSE_ITERATOR, 106
 - QL_SPECIALIZE_ITERATOR_TRAITS, 106
- itmProbability
 - QuantLib::BlackModel, 160
- KnuthUniformRng
 - QuantLib::KnuthUniformRng, 384
- kurtosis
 - QuantLib::GeneralStatistics, 330
 - QuantLib::IncrementalStatistics, 362
- Lagrange
 - QuantLib::CubicSpline, 231
- LecuyerUniformRng
 - QuantLib::LecuyerUniformRng, 396
- limitMacros
 - QL_EPSILON, 98
 - QL_MAX_DOUBLE, 98
 - QL_MAX_INT, 98
 - QL_MIN_DOUBLE, 98
 - QL_MIN_INT, 98
 - QL_MIN_POSITIVE_DOUBLE, 98
- LinearInterpolation
 - QuantLib::LinearInterpolation, 401
- Link
 - QuantLib::Link, 404
- linkTo
 - QuantLib::Link, 404
 - QuantLib::RelinkableHandle, 538
- localVolImpl
 - QuantLib::LocalVolCurve, 409
- LogLinearInterpolation
 - QuantLib::LogLinearInterpolation, 414
- macros
 - QL_DUMMY_RETURN, 95
 - QL_IO_INIT, 96
- Math functions, 97
- matrixSqrt
 - QuantLib::Matrix, 421

- max
 - QuantLib::GeneralStatistics, 331
 - QuantLib::IncrementalStatistics, 362
- mean
 - QuantLib::GeneralStatistics, 330
 - QuantLib::IncrementalStatistics, 361
- MersenneTwisterUniformRng
 - QuantLib::MersenneTwisterUniformRng, 443
- min
 - QuantLib::GeneralStatistics, 330
 - QuantLib::IncrementalStatistics, 362
- Min and max functions, 103
- MonotonicCubicSpline
 - QuantLib::MonotonicCubicSpline, 447
- name
 - QuantLib::Calendar, 185
 - QuantLib::DayCounter, 241
 - QuantLib::Index, 363
 - QuantLib::Xibor, 632
- NaturalCubicSpline
 - QuantLib::NaturalCubicSpline, 458
- NaturalMonotonicCubicSpline
 - QuantLib::NaturalMonotonicCubicSpline, 459
- next
 - QuantLib::KnuthUniformRng, 384
 - QuantLib::LecuyerUniformRng, 396
 - QuantLib::MersenneTwisterUniformRng, 443
- NotAKnot
 - QuantLib::CubicSpline, 231
- notifyObservers
 - QuantLib::Observable, 472
- Numeric limits, 98
- operator+=
 - QuantLib::Matrix, 420
- operator==
 - QuantLib::Calendar, 186
 - QuantLib::DayCounter, 241
- PathGenerator_old
 - QuantLib::PathGenerator_old, 503
- percentile
 - QuantLib::GeneralStatistics, 331
- performCalculations
 - QuantLib::BarrierOption, 137
 - QuantLib::BasketOption, 141
 - QuantLib::DiscreteAveragingAsianOption, 256
 - QuantLib::ForwardVanillaOption, 316
 - QuantLib::Instrument, 369
 - QuantLib::LazyObject, 392
 - QuantLib::MultiAssetOption, 452
 - QuantLib::OneAssetOption, 478
 - QuantLib::OneAssetStrikedOption, 482
 - QuantLib::QuantoVanillaOption, 531
 - QuantLib::Stock, 574
 - QuantLib::Swap, 582
 - QuantLib::Swaption, 585
 - QuantLib::VanillaOption, 624
- Periodic
 - QuantLib::CubicSpline, 231
- PiecewiseFlatForward
 - QuantLib::PiecewiseFlatForward, 512
- postAdjustValues
 - QuantLib::DiscretizedAsset, 260
 - QuantLib::DiscretizedOption, 263
- potentialUpside
 - QuantLib::GenericRiskStatistics, 335
- preAdjustValues
 - QuantLib::DiscretizedAsset, 260
- pseudoSqrt
 - QuantLib::Matrix, 420
- ql/argsandresults.hpp, 641
- ql/calendar.hpp, 642
- ql/Calendars/budapest.hpp, 643
- ql/Calendars/copenhagen.hpp, 644
- ql/Calendars/frankfurt.hpp, 645
- ql/Calendars/helsinki.hpp, 646
- ql/Calendars/johannesburg.hpp, 647
- ql/Calendars/jointcalendar.hpp, 648
- ql/Calendars/london.hpp, 649
- ql/Calendars/milan.hpp, 650
- ql/Calendars/newyork.hpp, 651
- ql/Calendars/nullcalendar.hpp, 652
- ql/Calendars/oslo.hpp, 653
- ql/Calendars/stockholm.hpp, 654
- ql/Calendars/sydney.hpp, 655
- ql/Calendars/target.hpp, 656
- ql/Calendars/tokyo.hpp, 657
- ql/Calendars/toronto.hpp, 658
- ql/Calendars/warsaw.hpp, 659
- ql/Calendars/wellington.hpp, 660
- ql/Calendars/zurich.hpp, 661
- ql/capvolstructures.hpp, 662
- ql/cashflow.hpp, 663
- ql/CashFlows/basispointsensitivity.hpp, 664
- ql/CashFlows/cashflowvectors.hpp, 665
- ql/CashFlows/coupon.hpp, 666
- ql/CashFlows/fixedratecoupon.hpp, 667
- ql/CashFlows/floatingratecoupon.hpp, 668
- ql/CashFlows/inarreindexedcoupon.hpp, 669

- ql/CashFlows/indexcashflowvectors.hpp, 670
- ql/CashFlows/indexedcoupon.hpp, 671
- ql/CashFlows/parcoupon.hpp, 672
- ql/CashFlows/shortfloatingcoupon.hpp, 673
- ql/CashFlows/shortindexedcoupon.hpp, 674
- ql/CashFlows/simplecashflow.hpp, 675
- ql/CashFlows/timebasket.hpp, 676
- ql/CashFlows/upfrontindexedcoupon.hpp, 677
- ql/currency.hpp, 678
- ql/dataformatters.hpp, 679
- ql/dataparsers.hpp, 680
- ql/date.hpp, 681
- ql/daycounter.hpp, 682
- ql/DayCounters/actual360.hpp, 683
- ql/DayCounters/actual365.hpp, 684
- ql/DayCounters/actualactual.hpp, 685
- ql/DayCounters/simpliedaycounter.hpp, 686
- ql/DayCounters/thirty360.hpp, 687
- ql/diffusionprocess.hpp, 688
- ql/discretizedasset.hpp, 689
- ql/disposable.hpp, 690
- ql/errors.hpp, 691
- ql/exercise.hpp, 693
- ql/FiniteDifferences/americancondition.hpp, 694
- ql/FiniteDifferences/boundarycondition.hpp, 695
- ql/FiniteDifferences/bsmoperator.hpp, 696
- ql/FiniteDifferences/cranknicolson.hpp, 697
- ql/FiniteDifferences/dminus.hpp, 698
- ql/FiniteDifferences/dplus.hpp, 699
- ql/FiniteDifferences/dplusdminus.hpp, 700
- ql/FiniteDifferences/dzero.hpp, 701
- ql/FiniteDifferences/expliciteuler.hpp, 702
- ql/FiniteDifferences/fdtypedefs.hpp, 703
- ql/FiniteDifferences/finitedifferencemodel.hpp, 704
- ql/FiniteDifferences/impliciteuler.hpp, 705
- ql/FiniteDifferences/mixedscheme.hpp, 706
- ql/FiniteDifferences/onefactoroperator.hpp, 707
- ql/FiniteDifferences/shoutcondition.hpp, 708
- ql/FiniteDifferences/stepcondition.hpp, 709
- ql/FiniteDifferences/tridiagonaloperator.hpp, 710
- ql/FiniteDifferences/valueatcenter.hpp, 711
- ql/functions/daycounters.hpp, 712
- ql/functions/mathf.hpp, 713
- ql/functions/vols.hpp, 714
- ql/grid.hpp, 715
- ql/handle.hpp, 716
- ql/history.hpp, 717
- ql/index.hpp, 718
- ql/Indexes/audlibor.hpp, 719
- ql/Indexes/cadlibor.hpp, 720
- ql/Indexes/chflibor.hpp, 721
- ql/Indexes/euribor.hpp, 722
- ql/Indexes/gbplibor.hpp, 723
- ql/Indexes/jpylibor.hpp, 724
- ql/Indexes/usdlibor.hpp, 725
- ql/Indexes/xibor.hpp, 726
- ql/Indexes/xibormanager.hpp, 727
- ql/Indexes/zarlibor.hpp, 728
- ql/instrument.hpp, 729
- ql/Instruments/asianooption.hpp, 730
- ql/Instruments/barrierooption.hpp, 731
- ql/Instruments/basketoption.hpp, 732
- ql/Instruments/capfloor.hpp, 733
- ql/Instruments/cliquetooption.hpp, 734
- ql/Instruments/forwardvanillaoption.hpp, 736
- ql/Instruments/multiassetoption.hpp, 737
- ql/Instruments/oneassetoption.hpp, 738
- ql/Instruments/oneassetstrikedoption.hpp, 739
- ql/Instruments/payoffs.hpp, 740
- ql/Instruments/quantoforwardvanillaoption.hpp, 741
- ql/Instruments/quantovanillaoption.hpp, 742
- ql/Instruments/simpleswap.hpp, 743
- ql/Instruments/stock.hpp, 744
- ql/Instruments/swap.hpp, 745
- ql/Instruments/swaption.hpp, 746
- ql/Instruments/vanillaoption.hpp, 747
- ql/Lattices/binomialtree.hpp, 748
- ql/Lattices/bsmlattice.hpp, 749
- ql/Lattices/lattice.hpp, 750
- ql/Lattices/lattice2d.hpp, 751
- ql/Lattices/tree.hpp, 752
- ql/Lattices/trinomialtree.hpp, 753
- ql/marketelement.hpp, 754
- ql/Math/array.hpp, 755
- ql/Math/beta.hpp, 756
- ql/Math/bicubicsplineinterpolation.hpp, 757
- ql/Math/bilinearinterpolation.hpp, 758
- ql/Math/binomialdistribution.hpp, 759
- ql/Math/bivariatenormaldistribution.hpp, 760
- ql/Math/chisquaredistribution.hpp, 761
- ql/Math/choleskydecomposition.hpp, 762
- ql/Math/comparison.hpp, 763
- ql/Math/cubicspline.hpp, 764
- ql/Math/discrepancystatistics.hpp, 765
- ql/Math/errorfunction.hpp, 766

- ql/Math/factorial.hpp, 767
- ql/Math/functional.hpp, 768
- ql/Math/gammadistribution.hpp, 769
- ql/Math/gaussianstatistics.hpp, 770
- ql/Math/generalstatistics.hpp, 771
- ql/Math/incompletestgamma.hpp, 772
- ql/Math/incrementalstatistics.hpp, 773
- ql/Math/interpolation.hpp, 774
- ql/Math/interpolation2D.hpp, 775
- ql/Math/interpolationtraits.hpp, 776
- ql/Math/kronrodintegral.hpp, 777
- ql/Math/lexicographicalview.hpp, 778
- ql/Math/linearinterpolation.hpp, 779
- ql/Math/loglinearinterpolation.hpp, 780
- ql/Math/matrix.hpp, 781
- ql/Math/normaldistribution.hpp, 782
- ql/Math/poissondistribution.hpp, 783
- ql/Math/primenumbers.hpp, 784
- ql/Math/pseudosqrt.hpp, 785
- ql/Math/riskstatistics.hpp, 786
- ql/Math/segmentintegral.hpp, 787
- ql/Math/sequencestatistics.hpp, 788
- ql/Math/simpsonintegral.hpp, 790
- ql/Math/statistics.hpp, 791
- ql/Math/svd.hpp, 792
- ql/Math/symmetriceigenvalues.hpp, 793
- ql/Math/symmetricschurdecomposition.hpp, 794
- ql/Math/trapezoidintegral.hpp, 795
- ql/MonteCarlo/brownianbridge.hpp, 796
- ql/MonteCarlo/getcovariance.hpp, 797
- ql/MonteCarlo/mctraits.hpp, 798
- ql/MonteCarlo/mctypedefs.hpp, 799
- ql/MonteCarlo/montecarlomodel.hpp, 800
- ql/MonteCarlo/multipath.hpp, 801
- ql/MonteCarlo/multipathgenerator.hpp, 802
- ql/MonteCarlo/path.hpp, 803
- ql/MonteCarlo/pathgenerator.hpp, 804
- ql/MonteCarlo/pathpricer.hpp, 805
- ql/MonteCarlo/sample.hpp, 806
- ql/null.hpp, 807
- ql/numericalmethod.hpp, 808
- ql/Optimization/armijo.hpp, 809
- ql/Optimization/conjugategradient.hpp, 810
- ql/Optimization/constraint.hpp, 811
- ql/Optimization/costfunction.hpp, 812
- ql/Optimization/criteria.hpp, 813
- ql/Optimization/leastsquare.hpp, 814
- ql/Optimization/linearssearch.hpp, 815
- ql/Optimization/method.hpp, 816
- ql/Optimization/problem.hpp, 817
- ql/Optimization/simplex.hpp, 818
- ql/Optimization/steepestdescent.hpp, 819
- ql/option.hpp, 820
- ql/Patterns/bridge.hpp, 821
- ql/Patterns/composite.hpp, 822
- ql/Patterns/curiouslyrecurring.hpp, 823
- ql/Patterns/lazyobject.hpp, 824
- ql/Patterns/observable.hpp, 825
- ql/Patterns/visitor.hpp, 826
- ql/payoff.hpp, 827
- ql/Pricers/cliquestoption.hpp, 735
- ql/Pricers/continuousgeometricapo.hpp, 828
- ql/Pricers/discretegeometricapo.hpp, 829
- ql/Pricers/discretegeometricaso.hpp, 830
- ql/Pricers/europeanoption.hpp, 831
- ql/Pricers/fdamericanoption.hpp, 832
- ql/Pricers/fdbermudanoption.hpp, 833
- ql/Pricers/fdbsmoption.hpp, 834
- ql/Pricers/fddividendamericanoption.hpp, 835
- ql/Pricers/fddividendeuropeanoption.hpp, 836
- ql/Pricers/fddividendoption.hpp, 837
- ql/Pricers/fddividendshoutoption.hpp, 838
- ql/Pricers/fdeuropean.hpp, 839
- ql/Pricers/fdmultiplieroption.hpp, 840
- ql/Pricers/fdshoutoption.hpp, 841
- ql/Pricers/fdstepconditionoption.hpp, 842
- ql/Pricers/mcbasket.hpp, 843
- ql/Pricers/mccliquestoption.hpp, 844
- ql/Pricers/mcdiscretearithmeticao.hpp, 845
- ql/Pricers/mcdiscretearithmeticaso.hpp, 846
- ql/Pricers/mceverest.hpp, 847
- ql/Pricers/mchimalaya.hpp, 848
- ql/Pricers/mcmaxbasket.hpp, 849
- ql/Pricers/mcpagoda.hpp, 850
- ql/Pricers/mcperformanceoption.hpp, 851
- ql/Pricers/mcpricer.hpp, 852
- ql/Pricers/performanceoption.hpp, 853
- ql/Pricers/singleassetoption.hpp, 854
- ql/pricingengine.hpp, 855
- ql/PricingEngines/americanpayoffatexpiry.hpp, 856
- ql/PricingEngines/americanpayoffathit.hpp, 857
- ql/PricingEngines/Asian/analyticasianengine.hpp, 858
- ql/PricingEngines/Barrier/analyticbarrierengine.hpp, 859
- ql/PricingEngines/Barrier/mcbarrierengine.hpp, 860
- ql/PricingEngines/Basket/mcamericanbasketengine.hpp, 861
- ql/PricingEngines/Basket/mcbasketengine.hpp, 862
- ql/PricingEngines/Basket/stulzengine.hpp, 863

- ql/PricingEngines/blackformula.hpp, 864
- ql/PricingEngines/blackmodel.hpp, 865
- ql/PricingEngines/CapFloor/analyticalcapfloor.hpp, 866
- ql/PricingEngines/CapFloor/blackcapfloor.hpp, 867
- ql/PricingEngines/CapFloor/capfloorpricer.hpp, 868
- ql/PricingEngines/CapFloor/treecapfloor.hpp, 869
- ql/PricingEngines/Forward/forwardengine.hpp, 870
- ql/PricingEngines/Forward/forwardperformanceengine.hpp, 871
- ql/PricingEngines/genericmodelengine.hpp, 872
- ql/PricingEngines/latticeshortratemodelengine.hpp, 873
- ql/PricingEngines/mcsimulation.hpp, 874
- ql/PricingEngines/Quanto/quantoengine.hpp, 875
- ql/PricingEngines/Swaption/blackswaption.hpp, 876
- ql/PricingEngines/Swaption/jamshidianswaption.hpp, 877
- ql/PricingEngines/Swaption/swaptionpricer.hpp, 878
- ql/PricingEngines/Swaption/treeswaption.hpp, 879
- ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp, 880
- ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp, 881
- ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp, 882
- ql/PricingEngines/Vanilla/binomialengine.hpp, 883
- ql/PricingEngines/Vanilla/bjerkstundstenslandengine.hpp, 884
- ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp, 885
- ql/PricingEngines/Vanilla/integralengine.hpp, 886
- ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp, 887
- ql/PricingEngines/Vanilla/mcdigitalengine.hpp, 888
- ql/PricingEngines/Vanilla/mceuropeanengine.hpp, 889
- ql/PricingEngines/Vanilla/mcvanillaengine.hpp, 890
- ql/qldefines.hpp, 891
- ql/RandomNumbers/boxmullergaussianrng.hpp, 893
- ql/RandomNumbers/centrallimitgaussianrng.hpp, 894
- ql/RandomNumbers/haltonrsg.hpp, 895
- ql/RandomNumbers/inversecumgaussianrng.hpp, 896
- ql/RandomNumbers/inversecumgaussianrsg.hpp, 897
- ql/RandomNumbers/knuthuniformrng.hpp, 898
- ql/RandomNumbers/lecuyeruniformrng.hpp, 899
- ql/RandomNumbers/mt19937uniformrng.hpp, 900
- ql/RandomNumbers/randomarraygenerator.hpp, 901
- ql/RandomNumbers/randomsequencegenerator.hpp, 902
- ql/RandomNumbers/rngtraits.hpp, 903
- ql/RandomNumbers/rngtypedefs.hpp, 904
- ql/RandomNumbers/sobolrsg.hpp, 905
- ql/relinkablehandle.hpp, 906
- ql/scheduler.hpp, 907
- ql/ShortRateModels/calibrationhelper.hpp, 908
- ql/ShortRateModels/CalibrationHelpers/caphelper.hpp, 909
- ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp, 910
- ql/ShortRateModels/model.hpp, 911
- ql/ShortRateModels/onefactormodel.hpp, 912
- ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp, 913
- ql/ShortRateModels/OneFactorModels/coxingersollross.hpp, 914
- ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp, 915
- ql/ShortRateModels/OneFactorModels/hullwhite.hpp, 916
- ql/ShortRateModels/OneFactorModels/vasicek.hpp, 917
- ql/ShortRateModels/parameter.hpp, 918
- ql/ShortRateModels/twofactormodel.hpp, 919
- ql/ShortRateModels/TwoFactorModels/g2.hpp, 920
- ql/solver1d.hpp, 921
- ql/Solvers1D/bisection.hpp, 922
- ql/Solvers1D/brent.hpp, 923
- ql/Solvers1D/falseposition.hpp, 924
- ql/Solvers1D/newton.hpp, 925
- ql/Solvers1D/newtonsafe.hpp, 926
- ql/Solvers1D/ridder.hpp, 927
- ql/Solvers1D/secant.hpp, 928

- ql/stochasticprocess.hpp, 929
- ql/swaptionvolstructure.hpp, 930
- ql/termstructure.hpp, 931
- ql/TermStructures/affinetermstructure.hpp, 932
- ql/TermStructures/compoundforward.hpp, 933
- ql/TermStructures/discountcurve.hpp, 934
- ql/TermStructures/drifttermstructure.hpp, 935
- ql/TermStructures/extendeddiscountcurve.hpp, 936
- ql/TermStructures/flatforward.hpp, 937
- ql/TermStructures/forwardspreadetermstructure.hpp, 938
- ql/TermStructures/IMPLIEDtermstructure.hpp, 939
- ql/TermStructures/piecewiseflatforward.hpp, 940
- ql/TermStructures/quantotermstructure.hpp, 941
- ql/TermStructures/ratehelpers.hpp, 942
- ql/TermStructures/zerocurve.hpp, 943
- ql/TermStructures/zerospreadetermstructure.hpp, 944
- ql/types.hpp, 945
- ql/Utilities/combiningiterator.hpp, 946
- ql/Utilities/couplingiterator.hpp, 947
- ql/Utilities/filteringiterator.hpp, 948
- ql/Utilities/iteratorcategories.hpp, 949
- ql/Utilities/processingiterator.hpp, 950
- ql/Utilities/steppingiterator.hpp, 951
- ql/Volatilities/blackconstantvol.hpp, 952
- ql/Volatilities/blackvariancecurve.hpp, 953
- ql/Volatilities/blackvariancesurface.hpp, 954
- ql/Volatilities/capflatvolvector.hpp, 955
- ql/Volatilities/IMPLIEDvoltermstructure.hpp, 956
- ql/Volatilities/localconstantvol.hpp, 957
- ql/Volatilities/localvolcurve.hpp, 958
- ql/Volatilities/localvolsurface.hpp, 959
- ql/Volatilities/swaptionvolmatrix.hpp, 960
- ql/voltermstructure.hpp, 961
- QL_ALLOW_TEMPLATE_METHOD_CALLS
 - templateMacros, 104
- QL_ASSERT
 - errors.hpp, 691
- QL_DECLARE_TEMPLATE_SPECIALIZATIONS
 - templateMacros, 104
- QL_DUMMY_RETURN
 - macros, 95
- QL_ENSURE
 - errors.hpp, 692
- QL_EPSILON
 - limitMacros, 98
- QL_FAIL
 - errors.hpp, 691
- QL_FULL_ITERATOR_SUPPORT
 - iteratorMacros, 106
- QL_IO_INIT
 - macros, 96
- QL_ITERATOR
 - iteratorMacros, 106
- QL_ITERATOR_TRAITS
 - iteratorMacros, 106
- QL_MAX_DOUBLE
 - limitMacros, 98
- QL_MAX_INT
 - limitMacros, 98
- QL_MIN_DOUBLE
 - limitMacros, 98
- QL_MIN_INT
 - limitMacros, 98
- QL_MIN_POSITIVE_DOUBLE
 - limitMacros, 98
- QL_REQUIRE
 - errors.hpp, 692
- QL_REVERSE_ITERATOR
 - iteratorMacros, 106
- QL_SPECIALIZE_ITERATOR_TRAITS
 - iteratorMacros, 106
- QL_TEMPLATE_-METAPROGRAMMING_WORKS
 - templateMacros, 104
- QL_TYPENAME
 - templateMacros, 104
- QuantLib::Actual360, 107
- QuantLib::Actual365, 108
- QuantLib::ActualActual, 109
- QuantLib::AcyclicVisitor, 110
- QuantLib::AdditiveEQPBinomialTree, 111
- QuantLib::AffineModel, 112
- QuantLib::AffineTermStructure, 113
- QuantLib::AffineTermStructure
 - update, 114
- QuantLib::AmericanCondition, 115
- QuantLib::AmericanExercise, 116
- QuantLib::AmericanPayoffAtExpiry, 117
- QuantLib::AmericanPayoffAtHit, 118
- QuantLib::AnalyticalCapFloor, 119
- QuantLib::AnalyticBarrierEngine, 120
- QuantLib::AnalyticDigitalAmericanEngine, 121
- QuantLib::AnalyticDiscreteAveragingAsianEngine, 122
- QuantLib::AnalyticEuropeanEngine, 123

- QuantLib::Arguments, 124
- QuantLib::ArmijoLineSearch, 125
- QuantLib::Array, 126
- QuantLib::ArrayFormatter, 128
- QuantLib::AssertionFailedError, 129
- QuantLib::AssetOrNothingPayoff, 130
- QuantLib::AUDLibor, 131
- QuantLib::Average, 132
- QuantLib::BaroneAdesiWhaleyApproximationEngine, 133
- QuantLib::Barrier, 134
- QuantLib::BarrierEngine, 135
- QuantLib::BarrierOption, 136
- QuantLib::BarrierOption
 - performCalculations, 137
 - setupArguments, 137
- QuantLib::BarrierOption::arguments, 138
- QuantLib::BasketEngine, 139
- QuantLib::BasketOption, 140
- QuantLib::BasketOption
 - performCalculations, 141
 - setupArguments, 140
- QuantLib::BasketOption::arguments, 142
- QuantLib::BermudanExercise, 143
- QuantLib::BicubicSpline, 144
- QuantLib::BicubicSpline
 - BicubicSpline, 144
- QuantLib::BicubicSpline::Impl, 145
- QuantLib::BilinearInterpolation, 146
- QuantLib::BilinearInterpolation
 - BilinearInterpolation, 146
- QuantLib::BilinearInterpolation::Impl, 147
- QuantLib::BinomialDistribution, 148
- QuantLib::BinomialTree, 149
- QuantLib::BinomialVanillaEngine, 150
- QuantLib::Bisection, 151
- QuantLib::BivariateCumulativeNormalDistribution, 152
- QuantLib::BjerkstrandStenslandApproximationEngine, 153
- QuantLib::BlackCapFloor, 154
- QuantLib::BlackConstantVol, 155
- QuantLib::BlackConstantVol
 - update, 156
- QuantLib::BlackKarasinski, 157
- QuantLib::BlackKarasinski::Dynamics, 158
- QuantLib::BlackModel, 159
- QuantLib::BlackModel
 - formula, 159
 - itmProbability, 160
 - update, 159
- QuantLib::BlackScholesLattice, 161
- QuantLib::BlackScholesProcess, 162
- QuantLib::BlackScholesProcess
 - diffusion, 162
- QuantLib::BlackSwaption, 163
- QuantLib::BlackVarianceCurve, 164
- QuantLib::BlackVarianceCurve
 - update, 165
- QuantLib::BlackVarianceSurface, 166
- QuantLib::BlackVarianceSurface
 - update, 167
- QuantLib::BlackVarianceTermStructure, 168
- QuantLib::BlackVarianceTermStructure
 - blackVolImpl, 168
- QuantLib::BlackVolatilityTermStructure, 169
- QuantLib::BlackVolatilityTermStructure
 - blackVarianceImpl, 169
- QuantLib::BlackVolTermStructure, 170
- QuantLib::BoundaryCondition, 172
- QuantLib::BoundaryCondition
 - applyAfterApplying, 172
 - applyAfterSolving, 172
 - applyBeforeApplying, 172
 - applyBeforeSolving, 172
 - setTime, 173
 - Side, 172
- QuantLib::BoundaryConstraint, 174
- QuantLib::BoxMullerGaussianRng, 175
- QuantLib::BoxMullerGaussianRng
 - BoxMullerGaussianRng, 175
- QuantLib::BPSBasketCalculator, 176
- QuantLib::BPSCalculator, 177
- QuantLib::Brent, 178
- QuantLib::Bridge, 179
- QuantLib::BrownianBridge, 180
- QuantLib::BSMOperator, 181
- QuantLib::Budapest, 182
- QuantLib::CADLibor, 183
- QuantLib::Calendar, 184
- QuantLib::Calendar
 - advance, 186
 - Calendar, 185
 - isBusinessDay, 185
 - isEndOfMonth, 185
 - isHoliday, 186
 - name, 185
 - operator==, 186
 - roll, 186
- QuantLib::Calendar::WesternImpl, 187
- QuantLib::CalendarImpl, 188
- QuantLib::CalibrationHelper, 189
- QuantLib::CalibrationHelper
 - update, 190
- QuantLib::CalibrationSet, 191
- QuantLib::Cap, 192
- QuantLib::CapFlatVolatilityStructure, 193
- QuantLib::CapFlatVolatilityVector, 194

- QuantLib::CapFloor, 195
- QuantLib::CapFloor
 - setupArguments, 196
- QuantLib::CapFloor::arguments, 197
- QuantLib::CapFloor::results, 198
- QuantLib::CapletForwardVolatilityStructure, 199
- QuantLib::CashFlow, 200
- QuantLib::CashFlow
 - amount, 200
- QuantLib::CashOrNothingPayoff, 201
- QuantLib::CHFLibor, 202
- QuantLib::CLGaussianRng, 203
- QuantLib::CLGaussianRng
 - CLGaussianRng, 203
- QuantLib::CliquetEngine, 204
- QuantLib::CliquetOption::arguments, 205
- QuantLib::CliquetOptionPricer, 206
- QuantLib::Collar, 207
- QuantLib::combining_iterator, 208
- QuantLib::Composite, 210
- QuantLib::CompositeConstraint, 211
- QuantLib::CompositeQuote, 212
- QuantLib::CompositeQuote
 - update, 212
- QuantLib::ConjugateGradient, 213
- QuantLib::ConstantParameter, 214
- QuantLib::Constraint, 215
- QuantLib::ConstraintImpl, 216
- QuantLib::ContinuousGeometricAPO, 217
- QuantLib::Copenhagen, 218
- QuantLib::CostFunction, 219
- QuantLib::coupling_iterator, 220
- QuantLib::Coupon, 222
- QuantLib::Coupon
 - Coupon, 223
- QuantLib::CoxIngersollRoss, 224
- QuantLib::CoxIngersollRoss::Dynamics, 226
- QuantLib::CoxRossRubinstein, 227
- QuantLib::CrankNicolson, 228
- QuantLib::Cubic, 229
- QuantLib::CubicSpline, 230
 - FirstDerivative, 231
 - Lagrange, 231
 - NotAKnot, 231
 - Periodic, 231
 - SecondDerivative, 231
- QuantLib::CubicSpline
 - BoundaryCondition, 231
 - CubicSpline, 231
- QuantLib::CumulativeBinomialDistribution, 232
- QuantLib::CumulativeNormalDistribution, 233
- QuantLib::CumulativePoissonDistribution, 234
- QuantLib::CuriouslyRecurringTemplate, 235
- QuantLib::CurrencyFormatter, 236
- QuantLib::Date, 237
- QuantLib::DateFormatter, 239
- QuantLib::DayCounter, 240
- QuantLib::DayCounter
 - DayCounter, 241
 - name, 241
 - operator==, 241
- QuantLib::DayCounterImpl, 242
- QuantLib::DepositRateHelper, 243
- QuantLib::DepositRateHelper
 - setTermStructure, 243
- QuantLib::DerivedQuote, 245
- QuantLib::DerivedQuote
 - update, 245
- QuantLib::DiffusionProcess, 246
- QuantLib::DiffusionProcess
 - diffusion, 246
 - expectation, 246
 - variance, 246
- QuantLib::DirichletBC, 248
- QuantLib::DirichletBC
 - setTime, 248
- QuantLib::DiscountCurve, 249
- QuantLib::DiscountStructure, 251
- QuantLib::DiscountStructure
 - compoundForwardImpl, 251
 - forwardImpl, 251
 - zeroYieldImpl, 251
- QuantLib::DiscrepancyStatistics, 253
- QuantLib::DiscreteAveragingAsianEngine, 254
- QuantLib::DiscreteAveragingAsianOption, 255
- QuantLib::DiscreteAveragingAsianOption
 - performCalculations, 256
 - setupArguments, 256
- QuantLib::DiscreteAveragingAsianOption::arguments, 257
- QuantLib::DiscreteGeometricAPO, 258
- QuantLib::DiscreteGeometricASO, 259
- QuantLib::DiscretizedAsset, 260
- QuantLib::DiscretizedAsset
 - postAdjustValues, 260
 - preAdjustValues, 260
- QuantLib::DiscretizedDiscountBond, 262
- QuantLib::DiscretizedOption, 263
- QuantLib::DiscretizedOption
 - postAdjustValues, 263
- QuantLib::Disposable, 264

- QuantLib::DMinus, 265
- QuantLib::DoubleFormatter, 266
- QuantLib::DPlus, 267
- QuantLib::DPlusDMinus, 268
- QuantLib::DriftTermStructure, 269
- QuantLib::DriftTermStructure
 - update, 270
- QuantLib::DZero, 271
- QuantLib::EarlyExercise, 272
- QuantLib::EndCriteria, 273
- QuantLib::EqualJumpsBinomialTree, 275
- QuantLib::EqualProbabilitiesBinomialTree, 276
- QuantLib::Error, 277
- QuantLib::Error
 - Error, 277
- QuantLib::ErrorFunction, 278
- QuantLib::Euribor, 279
- QuantLib::EuroFormatter, 280
- QuantLib::EuropeanExercise, 281
- QuantLib::EuropeanOption, 282
- QuantLib::Exercise, 283
- QuantLib::ExplicitEuler, 284
- QuantLib::ExtendedCoxIngersollRoss, 285
- QuantLib::ExtendedCoxIngersollRoss::Dynamics, 286
- QuantLib::ExtendedCoxIngersollRoss::FittingParameter, gaussianDownsideDeviation, 326
 - gaussianDownsideVariance, 326
 - gaussianExpectedShortfall, 327
 - gaussianPercentile, 327
 - gaussianPotentialUpside, 327
 - gaussianRegret, 327
 - gaussianValueAtRisk, 327
- QuantLib::ExtendedDiscountCurve, 288
- QuantLib::ExtendedDiscountCurve
 - compoundForwardImpl, 289
 - update, 289
- QuantLib::Factorial, 290
- QuantLib::FalsePosition, 291
- QuantLib::FdAmericanOption, 292
- QuantLib::FdBermudanOption, 293
- QuantLib::FdBsmOption, 294
- QuantLib::FdDividendAmericanOption, 296
- QuantLib::FdDividendEuropeanOption, 297
- QuantLib::FdDividendShoutOption, 298
- QuantLib::FdEuropean, 299
- QuantLib::FdStepConditionOption, 300
- QuantLib::filtering_iterator, 301
- QuantLib::FiniteDifferenceModel, 302
- QuantLib::FiniteDifferenceModel
 - rollback, 302
- QuantLib::FixedRateCoupon, 303
- QuantLib::FixedRateCoupon
 - amount, 304
- QuantLib::FloatingRateCoupon, 305
- QuantLib::Floor, 307
- QuantLib::ForwardEngine, 308
- QuantLib::ForwardOptionArguments, 309
- QuantLib::ForwardPerformanceEngine, 310
- QuantLib::ForwardRateStructure, 311
- QuantLib::ForwardRateStructure
 - compoundForwardImpl, 311
 - discountImpl, 311
 - zeroYieldImpl, 311
- QuantLib::ForwardSpreadedTermStructure, 313
- QuantLib::ForwardSpreadedTermStructure
 - update, 314
 - zeroYieldImpl, 314
- QuantLib::ForwardVanillaOption, 315
- QuantLib::ForwardVanillaOption
 - performCalculations, 316
 - setupArguments, 316
- QuantLib::Frankfurt, 317
- QuantLib::FraRateHelper, 318
- QuantLib::FraRateHelper
 - setTermStructure, 318
- QuantLib::FuturesRateHelper, 320
- QuantLib::G2, 321
- QuantLib::G2::FittingParameter, 323
- QuantLib::GammaFunction, 324
- QuantLib::GapPayoff, 325
- QuantLib::GaussianStatistics, 326
- QuantLib::GaussianStatistics
 - gaussianDownsideDeviation, 326
 - gaussianDownsideVariance, 326
 - gaussianExpectedShortfall, 327
 - gaussianPercentile, 327
 - gaussianPotentialUpside, 327
 - gaussianRegret, 327
 - gaussianValueAtRisk, 327
- QuantLib::GBPLibor, 328
- QuantLib::GeneralStatistics, 329
- QuantLib::GeneralStatistics
 - add, 331
 - errorEstimate, 330
 - expectationValue, 331
 - kurtosis, 330
 - max, 331
 - mean, 330
 - min, 330
 - percentile, 331
 - skewness, 330
 - standardDeviation, 330
 - topPercentile, 331
 - variance, 330
- QuantLib::GenericEngine, 332
- QuantLib::GenericModelEngine, 333
- QuantLib::GenericModelEngine
 - update, 333
- QuantLib::GenericRiskStatistics, 334
- QuantLib::GenericRiskStatistics

- averageShortfall, 336
- downsideDeviation, 335
- downsideVariance, 334
- expectedShortfall, 335
- potentialUpside, 335
- regret, 335
- semiDeviation, 334
- semiVariance, 334
- shortfall, 335
- valueAtRisk, 335
- QuantLib::Greeks, 337
- QuantLib::HaltonRsg, 338
- QuantLib::Handle, 339
- QuantLib::Handle
 - Handle, 340
- QuantLib::Helsinki, 341
- QuantLib::History, 342
- QuantLib::History
 - History, 343
- QuantLib::History::const_iterator, 345
- QuantLib::History::Entry, 346
- QuantLib::HullWhite, 347
- QuantLib::HullWhite::Dynamics, 348
- QuantLib::HullWhite::FittingParameter, 349
- QuantLib::ICGaussianRng, 350
- QuantLib::ICGaussianRng
 - ICGaussianRng, 350
- QuantLib::ICGaussianRsg, 351
- QuantLib::IllegalArgumentError, 352
- QuantLib::IllegalResultError, 353
- QuantLib::ImplicitEuler, 354
- QuantLib::ImpliedTermStructure, 355
- QuantLib::ImpliedTermStructure
 - update, 356
- QuantLib::ImpliedVolTermStructure, 357
- QuantLib::ImpliedVolTermStructure
 - update, 358
- QuantLib::InArrearIndexedCoupon, 359
- QuantLib::IncrementalStatistics, 360
- QuantLib::IncrementalStatistics
 - add, 362
 - addSequence, 362
 - downsideDeviation, 361
 - downsideVariance, 361
 - errorEstimate, 361
 - kurtosis, 362
 - max, 362
 - mean, 361
 - min, 362
 - skewness, 362
 - standardDeviation, 361
 - variance, 361
- QuantLib::Index, 363
- QuantLib::Index
 - fixing, 363
 - name, 363
- QuantLib::IndexedCoupon, 364
- QuantLib::IndexedCoupon
 - amount, 365
 - update, 365
- QuantLib::IndexError, 366
- QuantLib::Instrument, 367
- QuantLib::Instrument
 - calculate, 368
 - performCalculations, 369
 - setPricingEngine, 368
 - setupArguments, 368
 - setupExpired, 368
- QuantLib::IntegerFormatter, 370
- QuantLib::IntegralEngine, 371
- QuantLib::Interpolation, 372
- QuantLib::Interpolation2D, 373
- QuantLib::Interpolation2D::templateImpl, 374
- QuantLib::Interpolation2DImpl, 375
- QuantLib::Interpolation::templateImpl, 376
- QuantLib::InterpolationImpl, 377
- QuantLib::InverseCumulativeNormal, 378
- QuantLib::JamshidianSwaption, 379
- QuantLib::JarrowRudd, 380
- QuantLib::Johannesburg, 381
- QuantLib::JointCalendar, 382
- QuantLib::JPYLibor, 383
- QuantLib::KnuthUniformRng, 384
- QuantLib::KnuthUniformRng
 - KnuthUniformRng, 384
 - next, 384
- QuantLib::KronrodIntegral, 385
- QuantLib::Lattice, 386
- QuantLib::Lattice
 - rollAlmostBack, 387
 - rollback, 387
- QuantLib::Lattice2D, 388
- QuantLib::LatticeShortRateModelEngine, 389
- QuantLib::LatticeShortRateModelEngine
 - update, 389
- QuantLib::LazyObject, 391
- QuantLib::LazyObject
 - calculate, 392
 - freeze, 392
 - performCalculations, 392
 - recalculate, 392
 - unfreeze, 392
 - update, 392
- QuantLib::LeastSquareFunction, 394
- QuantLib::LeastSquareProblem, 395
- QuantLib::LeastSquareProblem

- targetValueAndGradient, 395
- QuantLib::LecuyerUniformRng, 396
- QuantLib::LecuyerUniformRng
 - LecuyerUniformRng, 396
 - next, 396
- QuantLib::LeisenReimer, 397
- QuantLib::LexicographicalView, 398
- QuantLib::Linear, 400
- QuantLib::LinearInterpolation, 401
- QuantLib::LinearInterpolation
 - LinearInterpolation, 401
- QuantLib::LineSearch, 402
- QuantLib::Link, 404
- QuantLib::Link
 - Link, 404
 - linkTo, 404
- QuantLib::LocalConstantVol, 406
- QuantLib::LocalConstantVol
 - update, 407
- QuantLib::LocalVolCurve, 408
- QuantLib::LocalVolCurve
 - localVolImpl, 409
 - update, 409
- QuantLib::LocalVolSurface, 410
- QuantLib::LocalVolSurface
 - update, 411
- QuantLib::LocalVolTermStructure, 412
- QuantLib::LogLinear, 413
- QuantLib::LogLinearInterpolation, 414
- QuantLib::LogLinearInterpolation
 - LogLinearInterpolation, 414
- QuantLib::London, 415
- QuantLib::lowest_category_iterator, 416
- QuantLib::MakeSchedule, 417
- QuantLib::Matrix, 418
- QuantLib::Matrix
 - matrixSqrt, 421
 - operator+=, 420
 - pseudoSqrt, 420
 - rankReducedSqrt, 420
- QuantLib::MCAmericanBasketEngine, 422
- QuantLib::MCBarrierEngine, 423
- QuantLib::McBasket, 425
- QuantLib::MCBasketEngine, 426
- QuantLib::McCliquetOption, 428
- QuantLib::MCDigitalEngine, 429
- QuantLib::McDiscreteArithmeticAPO, 430
- QuantLib::McDiscreteArithmeticASO, 431
- QuantLib::MCEuropeanEngine, 432
- QuantLib::McEverest, 433
- QuantLib::McHimalaya, 434
- QuantLib::McMaxBasket, 435
- QuantLib::McPagoda, 436
- QuantLib::McPerformanceOption, 437
- QuantLib::McPricer, 438
- QuantLib::McSimulation, 439
- QuantLib::MCVanillaEngine, 441
- QuantLib::MersenneTwisterUniformRng, 443
 - MersenneTwisterUniformRng, 443
 - next, 443
- QuantLib::Milan, 444
- QuantLib::MixedScheme, 445
- QuantLib::MonotonicCubicSpline, 447
- QuantLib::MonotonicCubicSpline
 - MonotonicCubicSpline, 447
- QuantLib::MonteCarloModel, 448
- QuantLib::MoreGreeks, 449
- QuantLib::MoroInverseCumulativeNormal, 450
- QuantLib::MultiAssetOption, 451
- QuantLib::MultiAssetOption
 - performCalculations, 452
 - setupArguments, 452
 - setupExpired, 452
- QuantLib::MultiAssetOption::arguments, 453
- QuantLib::MultiAssetOption::results, 454
- QuantLib::MultiPath, 455
- QuantLib::MultiPathGenerator, 456
- QuantLib::MultiPathGenerator_old, 457
- QuantLib::NaturalCubicSpline, 458
- QuantLib::NaturalCubicSpline
 - NaturalCubicSpline, 458
- QuantLib::NaturalMonotonicCubicSpline, 459
- QuantLib::NaturalMonotonicCubicSpline
 - NaturalMonotonicCubicSpline, 459
- QuantLib::NeumannBC, 460
- QuantLib::NeumannBC
 - setTime, 460
- QuantLib::Newton, 461
- QuantLib::NewtonSafe, 462
- QuantLib::NewYork, 463
- QuantLib::NoConstraint, 464
- QuantLib::NonLinearLeastSquare, 465
- QuantLib::NormalDistribution, 466
- QuantLib::Null, 467
- QuantLib::NullCalendar, 468
- QuantLib::NullParameter, 469
- QuantLib::NumericalMethod, 470
- QuantLib::Observable, 471
- QuantLib::Observable
 - notifyObservers, 472
- QuantLib::Observer, 473
- QuantLib::Observer
 - update, 475

- QuantLib::OneAssetOption, 476
- QuantLib::OneAssetOption
 - impliedVolatility, 477
 - performCalculations, 478
 - setupArguments, 477
 - setupExpired, 478
- QuantLib::OneAssetOption::arguments, 479
- QuantLib::OneAssetOption::results, 480
- QuantLib::OneAssetStrikedOption, 481
- QuantLib::OneAssetStrikedOption
 - performCalculations, 482
 - setupArguments, 482
- QuantLib::OneFactorAffineModel, 483
- QuantLib::OneFactorModel, 484
- QuantLib::OneFactorModel::ShortRateDynamics, 485
- QuantLib::OneFactorModel::ShortRateTree, 486
- QuantLib::OneFactorOperator, 487
- QuantLib::OptimizationMethod, 488
- QuantLib::Option, 490
- QuantLib::Option::arguments, 491
- QuantLib::OptionTypeFormatter, 492
- QuantLib::OrnsteinUhlenbeckProcess, 493
- QuantLib::OrnsteinUhlenbeckProcess
 - diffusion, 493
 - expectation, 493
 - variance, 493
- QuantLib::Oslo, 495
- QuantLib::OutOfMemoryError, 496
- QuantLib::Parameter, 497
- QuantLib::ParameterImpl, 498
- QuantLib::ParCoupon, 499
- QuantLib::ParCoupon
 - amount, 500
 - update, 500
- QuantLib::Path, 501
- QuantLib::PathGenerator, 502
- QuantLib::PathGenerator_old, 503
- QuantLib::PathGenerator_old
 - PathGenerator_old, 503
- QuantLib::PathPricer, 504
- QuantLib::PathPricer_old, 505
- QuantLib::Payoff, 506
- QuantLib::PercentageStrikePayoff, 507
- QuantLib::PerformanceOption, 508
- QuantLib::Period, 509
- QuantLib::PiecewiseConstantParameter, 510
- QuantLib::PiecewiseFlatForward, 511
- QuantLib::PiecewiseFlatForward
 - PiecewiseFlatForward, 512
- QuantLib::PlainVanillaPayoff, 513
- QuantLib::PoissonDistribution, 514
- QuantLib::PositiveConstraint, 515
- QuantLib::PostconditionNotSatisfiedError, 516
- QuantLib::PreconditionNotSatisfiedError, 517
- QuantLib::PricingEngine, 518
- QuantLib::PrimeNumbers, 519
- QuantLib::Problem, 520
- QuantLib::processing_iterator, 521
- QuantLib::QuantoEngine, 523
- QuantLib::QuantoForwardVanillaOption, 524
- QuantLib::QuantoForwardVanillaOption
 - setupArguments, 525
- QuantLib::QuantoOptionArguments, 526
- QuantLib::QuantoOptionResults, 527
- QuantLib::QuantoTermStructure, 528
- QuantLib::QuantoTermStructure
 - update, 529
- QuantLib::QuantoVanillaOption, 530
- QuantLib::QuantoVanillaOption
 - performCalculations, 531
 - setupArguments, 531
 - setupExpired, 531
- QuantLib::Quote, 532
- QuantLib::RandomArrayGenerator, 533
- QuantLib::RandomSequenceGenerator, 534
- QuantLib::RateFormatter, 535
- QuantLib::RateHelper, 536
- QuantLib::RateHelper
 - setTermStructure, 537
 - update, 537
- QuantLib::RelinkableHandle, 538
- QuantLib::RelinkableHandle
 - linkTo, 538
 - RelinkableHandle, 538
- QuantLib::Results, 539
- QuantLib::Ridder, 540
- QuantLib::SalvagingAlgorithm, 541
- QuantLib::Sample, 542
- QuantLib::Schedule, 543
- QuantLib::Secant, 544
- QuantLib::SegmentIntegral, 545
- QuantLib::SequenceStatistics, 546
- QuantLib::Short, 548
- QuantLib::Short
 - amount, 548
- QuantLib::ShortFloatingRateCoupon, 549
- QuantLib::ShortRateModel, 550
- QuantLib::ShortRateModel
 - calibrate, 551
 - update, 551
- QuantLib::ShoutCondition, 552
- QuantLib::SimpleCashFlow, 553

- QuantLib::SimpleCashFlow
 - amount, [553](#)
- QuantLib::SimpleDayCounter, [554](#)
- QuantLib::SimpleQuote, [555](#)
- QuantLib::SimpleSwap, [556](#)
- QuantLib::SimpleSwap
 - setupArguments, [557](#)
- QuantLib::SimpleSwap::arguments, [558](#)
- QuantLib::SimpleSwap::results, [559](#)
- QuantLib::Simplex, [560](#)
- QuantLib::Simplex
 - Simplex, [560](#)
- QuantLib::SimpsonIntegral, [561](#)
- QuantLib::SingleAssetOption, [562](#)
- QuantLib::SingleAssetOption
 - impliedVolatility, [563](#)
- QuantLib::SobolRsg, [564](#)
- QuantLib::Solver1D, [565](#)
- QuantLib::Solver1D
 - setMaxEvaluations, [566](#)
 - solve, [566](#)
- QuantLib::SquareRootProcess, [567](#)
- QuantLib::SquareRootProcess
 - diffusion, [567](#)
- QuantLib::StatsHolder, [568](#)
- QuantLib::SteepestDescent, [569](#)
- QuantLib::StepCondition, [570](#)
- QuantLib::stepping_iterator, [571](#)
- QuantLib::StochasticProcess, [573](#)
- QuantLib::StochasticProcess
 - update, [573](#)
- QuantLib::Stock, [574](#)
- QuantLib::Stock
 - performCalculations, [574](#)
- QuantLib::Stockholm, [575](#)
- QuantLib::StrikedTypePayoff, [576](#)
- QuantLib::StringFormatter, [577](#)
- QuantLib::StulzEngine, [578](#)
- QuantLib::SuperSharePayoff, [579](#)
- QuantLib::SVD, [580](#)
- QuantLib::Swap, [581](#)
- QuantLib::Swap
 - performCalculations, [582](#)
 - sensitivity, [582](#)
 - setupExpired, [582](#)
- QuantLib::SwapRateHelper, [583](#)
- QuantLib::SwapRateHelper
 - setTermStructure, [584](#)
- QuantLib::Swaption, [585](#)
- QuantLib::Swaption
 - performCalculations, [585](#)
 - setupArguments, [585](#)
- QuantLib::Swaption::arguments, [587](#)
- QuantLib::Swaption::results, [588](#)
- QuantLib::SwaptionVolatilityMatrix, [589](#)
- QuantLib::SwaptionVolatilityStructure, [590](#)
- QuantLib::Sydney, [591](#)
- QuantLib::SymmetricSchurDecomposition, [592](#)
- QuantLib::TARGET, [593](#)
- QuantLib::TermStructure, [594](#)
- QuantLib::TermStructureConsistentModel, [596](#)
- QuantLib::TermStructureFittingParameter, [597](#)
- QuantLib::Thirty360, [598](#)
- QuantLib::Tian, [599](#)
- QuantLib::TimeBasket, [600](#)
- QuantLib::TimeGrid, [601](#)
- QuantLib::TimeGrid
 - TimeGrid, [601](#)
- QuantLib::Tokyo, [602](#)
- QuantLib::Toronto, [604](#)
- QuantLib::TrapezoidIntegral, [605](#)
- QuantLib::Tree, [607](#)
- QuantLib::TreeCapFloor, [608](#)
- QuantLib::TreeSwaption, [609](#)
- QuantLib::TridiagonalOperator, [610](#)
- QuantLib::TridiagonalOperator::TimeSetter, [612](#)
- QuantLib::Trigeorgis, [613](#)
- QuantLib::TrinomialBranching, [614](#)
- QuantLib::TrinomialTree, [615](#)
- QuantLib::TwoFactorModel, [616](#)
- QuantLib::TwoFactorModel::ShortRateDynamics, [617](#)
- QuantLib::TwoFactorModel::ShortRateTree, [618](#)
- QuantLib::TypePayoff, [619](#)
- QuantLib::UpFrontIndexedCoupon, [620](#)
- QuantLib::USDLibor, [621](#)
- QuantLib::Value, [622](#)
- QuantLib::VanillaEngine, [623](#)
- QuantLib::VanillaOption, [624](#)
- QuantLib::VanillaOption
 - performCalculations, [624](#)
- QuantLib::Vasicek, [626](#)
- QuantLib::Vasicek::Dynamics, [627](#)
- QuantLib::Visitor, [628](#)
- QuantLib::Warsaw, [629](#)
- QuantLib::Wellington, [630](#)
- QuantLib::Xibor, [631](#)
- QuantLib::Xibor
 - fixing, [632](#)
 - name, [632](#)
 - update, [632](#)
- QuantLib::XiborManager, [633](#)
- QuantLib::ZARLibor, [634](#)

- QuantLib::ZeroCurve, 635
- QuantLib::ZeroSpreadedTermStructure, 636
- QuantLib::ZeroSpreadedTermStructure
 - forwardImpl, 637
 - update, 637
- QuantLib::ZeroYieldStructure, 638
- QuantLib::ZeroYieldStructure
 - compoundForwardImpl, 638
 - discountImpl, 638
 - forwardImpl, 638
- QuantLib::Zurich, 640
- rankReducedSqrt
 - QuantLib::Matrix, 420
- recalculate
 - QuantLib::LazyObject, 392
- regret
 - QuantLib::GenericRiskStatistics, 335
- RelinkableHandle
 - QuantLib::RelinkableHandle, 538
- roll
 - QuantLib::Calendar, 186
- rollAlmostBack
 - QuantLib::Lattice, 387
- rollback
 - QuantLib::FiniteDifferenceModel, 302
 - QuantLib::Lattice, 387
- SecondDerivative
 - QuantLib::CubicSpline, 231
- semiDeviation
 - QuantLib::GenericRiskStatistics, 334
- semiVariance
 - QuantLib::GenericRiskStatistics, 334
- sensitivity
 - QuantLib::Swap, 582
- sequencestatistics.hpp
 - DEFINE_SEQUENCE_STAT_CONST_-METHOD_DOUBLE, 788
 - DEFINE_SEQUENCE_STAT_CONST_-METHOD_VOID, 788
- setMaxEvaluations
 - QuantLib::Solver1D, 566
- setPricingEngine
 - QuantLib::Instrument, 368
- setTermStructure
 - QuantLib::DepositRateHelper, 243
 - QuantLib::FraRateHelper, 318
 - QuantLib::RateHelper, 537
 - QuantLib::SwapRateHelper, 584
- setTime
 - QuantLib::BoundaryCondition, 173
 - QuantLib::DirichletBC, 248
 - QuantLib::NeumannBC, 460
- setupArguments
 - QuantLib::BarrierOption, 137
 - QuantLib::BasketOption, 140
 - QuantLib::CapFloor, 196
 - QuantLib::DiscreteAveragingAsian-Option, 256
 - QuantLib::ForwardVanillaOption, 316
 - QuantLib::Instrument, 368
 - QuantLib::MultiAssetOption, 452
 - QuantLib::OneAssetOption, 477
 - QuantLib::OneAssetStrikedOption, 482
 - QuantLib::QuantoForwardVanilla-Option, 525
 - QuantLib::QuantoVanillaOption, 531
 - QuantLib::SimpleSwap, 557
 - QuantLib::Swaption, 585
- setupExpired
 - QuantLib::Instrument, 368
 - QuantLib::MultiAssetOption, 452
 - QuantLib::OneAssetOption, 478
 - QuantLib::QuantoVanillaOption, 531
 - QuantLib::Swap, 582
- shortfall
 - QuantLib::GenericRiskStatistics, 335
- Side
 - QuantLib::BoundaryCondition, 172
- Simplex
 - QuantLib::Simplex, 560
- skewness
 - QuantLib::GeneralStatistics, 330
 - QuantLib::IncrementalStatistics, 362
- solve
 - QuantLib::Solver1D, 566
- standardDeviation
 - QuantLib::GeneralStatistics, 330
 - QuantLib::IncrementalStatistics, 361
- String functions, 100
- targetValueAndGradient
 - QuantLib::LeastSquareProblem, 395
- Template capabilities, 104
- templateMacros
 - QL_ALLOW_TEMPLATE_METHOD_-CALLS, 104
 - QL_DECLARE_TEMPLATE_-SPECIALIZATIONS, 104
 - QL_TEMPLATE_-METAPROGRAMMING_WORKS, 104
 - QL_TYPENAME, 104
- Time functions, 99
- TimeGrid
 - QuantLib::TimeGrid, 601
- topPercentile

- QuantLib::GeneralStatistics, [331](#)
- unfreeze
 - QuantLib::LazyObject, [392](#)
- update
 - QuantLib::AffineTermStructure, [114](#)
 - QuantLib::BlackConstantVol, [156](#)
 - QuantLib::BlackModel, [159](#)
 - QuantLib::BlackVarianceCurve, [165](#)
 - QuantLib::BlackVarianceSurface, [167](#)
 - QuantLib::CalibrationHelper, [190](#)
 - QuantLib::CompositeQuote, [212](#)
 - QuantLib::DerivedQuote, [245](#)
 - QuantLib::DriftTermStructure, [270](#)
 - QuantLib::ExtendedDiscountCurve, [289](#)
 - QuantLib::ForwardSpreadedTermStructure, [314](#)
 - QuantLib::GenericModelEngine, [333](#)
 - QuantLib::ImpliedTermStructure, [356](#)
 - QuantLib::ImpliedVolTermStructure, [358](#)
 - QuantLib::IndexedCoupon, [365](#)
 - QuantLib::LatticeShortRateModelEngine, [389](#)
 - QuantLib::LazyObject, [392](#)
 - QuantLib::LocalConstantVol, [407](#)
 - QuantLib::LocalVolCurve, [409](#)
 - QuantLib::LocalVolSurface, [411](#)
 - QuantLib::Observer, [475](#)
 - QuantLib::ParCoupon, [500](#)
 - QuantLib::QuantoTermStructure, [529](#)
 - QuantLib::RateHelper, [537](#)
 - QuantLib::ShortRateModel, [551](#)
 - QuantLib::StochasticProcess, [573](#)
 - QuantLib::Xibor, [632](#)
 - QuantLib::ZeroSpreadedTermStructure, [637](#)
- valueAtRisk
 - QuantLib::GenericRiskStatistics, [335](#)
- variance
 - QuantLib::DiffusionProcess, [246](#)
 - QuantLib::GeneralStatistics, [330](#)
 - QuantLib::IncrementalStatistics, [361](#)
 - QuantLib::OrnsteinUhlenbeckProcess, [493](#)
- zeroYieldImpl
 - QuantLib::DiscountStructure, [251](#)
 - QuantLib::ForwardRateStructure, [311](#)
 - QuantLib::ForwardSpreadedTermStructure, [314](#)