

QuantLib

An open source library for quantitative finance

Version 0.3.8

Generated by Doxygen 1.3.9.1

10 Dec 2004

Contents

1	Getting started	1
1.1	Introduction	1
1.2	Project overview	2
1.3	Where to get QuantLib	11
1.4	Installation	12
1.5	User configuration	13
1.6	Usage	14
1.7	Frequently asked questions	15
1.8	Version history	18
1.9	Additional resources	34
1.10	The QuantLib Group	35
1.11	QuantLib License	36
2	QuantLib Module Index	37
2.1	QuantLib Modules	37
3	QuantLib Hierarchical Index	39
3.1	QuantLib Class Hierarchy	39
4	QuantLib Class Index	53
4.1	QuantLib Class List	53
5	QuantLib File Index	65
5.1	QuantLib File List	65
6	QuantLib Module Documentation	75
6.1	Numeric types	75
6.2	Currencies and FX rates	77
6.3	Date and time calculations	80
6.4	Calendars	83

6.5	Day counters	85
6.6	Pricing engines	86
6.7	Asian option engines	87
6.8	Barrier option engines	88
6.9	Basket option engines	89
6.10	Cap/floor engines	90
6.11	Cliquet option engines	91
6.12	Forward option engines	92
6.13	Quanto option engines	93
6.14	Swaption engines	94
6.15	Vanilla option engines	95
6.16	Finite-differences framework	96
6.17	Short-rate modelling framework	102
6.18	Financial instruments	105
6.19	Lattice methods	108
6.20	Math tools	111
6.21	Monte Carlo framework	113
6.22	Design patterns	119
6.23	Term structures	120
6.24	Utilities	121
6.25	QuantLib macros	123
6.26	Generic macros	124
6.27	Math functions	125
6.28	Numeric limits	126
6.29	Time functions	127
6.30	String functions	128
6.31	Character functions	129
6.32	Min and max functions	130
6.33	Template capabilities	131
6.34	Iterator support	132
7	QuantLib Class Documentation	133
7.1	Actual360 Class Reference	133
7.2	Actual365Fixed Class Reference	134
7.3	ActualActual Class Reference	135
7.4	AcyclicVisitor Class Reference	136
7.5	AdditiveEQPBinomialTree Class Reference	137

7.6	AffineModel Class Reference	138
7.7	AffineTermStructure Class Reference	139
7.8	AmericanCondition Class Reference	141
7.9	AmericanExercise Class Reference	142
7.10	AmericanPayoffAtExpiry Class Reference	143
7.11	AmericanPayoffAtHit Class Reference	144
7.12	AnalyticBarrierEngine Class Reference	145
7.13	AnalyticCapFloorEngine Class Reference	146
7.14	AnalyticCliquetEngine Class Reference	147
7.15	AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference	148
7.16	AnalyticDigitalAmericanEngine Class Reference	149
7.17	AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference	150
7.18	AnalyticDividendEuropeanEngine Class Reference	151
7.19	AnalyticEuropeanEngine Class Reference	152
7.20	AnalyticPerformanceEngine Class Reference	153
7.21	Arguments Class Reference	154
7.22	ArmijoLineSearch Class Reference	155
7.23	Array Class Reference	156
7.24	ArrayFormatter Class Reference	159
7.25	ARSCurrency Class Reference	160
7.26	AssetOrNothingPayoff Class Reference	161
7.27	ATSCurrency Class Reference	162
7.28	AUDCurrency Class Reference	163
7.29	AUDLibor Class Reference	164
7.30	Average Struct Reference	165
7.31	BaroneAdesiWhaleyApproximationEngine Class Reference	166
7.32	Barrier Struct Reference	167
7.33	BarrierOption Class Reference	168
7.34	BarrierOption::arguments Class Reference	170
7.35	BarrierOption::engine Class Reference	171
7.36	BaseTermStructure Class Reference	172
7.37	BasketOption Class Reference	175
7.38	BasketOption::arguments Class Reference	176
7.39	BasketOption::engine Class Reference	177
7.40	BDTCurrency Class Reference	178
7.41	BEFCurrency Class Reference	179

7.42	Beijing Class Reference	180
7.43	BermudanExercise Class Reference	181
7.44	BGLCurrency Class Reference	182
7.45	BicubicSpline Class Reference	183
7.46	BilinearInterpolation Class Reference	184
7.47	BinomialDistribution Class Reference	185
7.48	BinomialTree Class Reference	186
7.49	BinomialVanillaEngine Class Template Reference	187
7.50	Bisection Class Reference	188
7.51	BivariateCumulativeNormalDistribution Class Reference	189
7.52	BjerkstrandStenslandApproximationEngine Class Reference	190
7.53	BlackCapFloorEngine Class Reference	191
7.54	BlackConstantVol Class Reference	192
7.55	BlackFormula Class Reference	194
7.56	BlackKarasinski Class Reference	195
7.57	BlackKarasinski::Dynamics Class Reference	196
7.58	BlackModel Class Reference	197
7.59	BlackScholesLattice Class Reference	199
7.60	BlackScholesProcess Class Reference	200
7.61	BlackSwaptionEngine Class Reference	202
7.62	BlackVarianceCurve Class Reference	203
7.63	BlackVarianceSurface Class Reference	205
7.64	BlackVarianceTermStructure Class Reference	207
7.65	BlackVolatilityTermStructure Class Reference	209
7.66	BlackVolTermStructure Class Reference	211
7.67	Bond Class Reference	214
7.68	BoundaryCondition Class Template Reference	217
7.69	BoundaryConstraint Class Reference	219
7.70	BoxMullerGaussianRng Class Template Reference	220
7.71	BPSBasketCalculator Class Reference	221
7.72	BPSCalculator Class Reference	222
7.73	Brent Class Reference	223
7.74	Bridge Class Template Reference	224
7.75	BRLCurrency Class Reference	225
7.76	BrownianBridge Class Template Reference	226
7.77	BSMOperator Class Reference	227

7.78	Budapest Class Reference	228
7.79	BYRCurrency Class Reference	229
7.80	CADCurrency Class Reference	230
7.81	CADLibor Class Reference	231
7.82	Calendar Class Reference	232
7.83	Calendar::WesternImpl Class Reference	236
7.84	CalendarImpl Class Reference	237
7.85	CalibrationHelper Class Reference	238
7.86	Cap Class Reference	240
7.87	CapFloor Class Reference	241
7.88	CapFloor::arguments Class Reference	243
7.89	CapFloor::results Class Reference	244
7.90	CapletConstantVolatility Class Reference	245
7.91	CapletVolatilityStructure Class Reference	246
7.92	CapVolatilityStructure Class Reference	248
7.93	CapVolatilityVector Class Reference	250
7.94	CashFlow Class Reference	252
7.95	CashOrNothingPayoff Class Reference	253
7.96	CeilingTruncation Class Reference	254
7.97	CHFCurrency Class Reference	255
7.98	CHFLibor Class Reference	256
7.99	CLGaussianRng Class Template Reference	257
7.100	CliquetOption Class Reference	258
7.101	CliquetOption::arguments Class Reference	260
7.102	CliquetOption::engine Class Reference	261
7.103	ClosestRounding Class Reference	262
7.104	CLPCurrency Class Reference	263
7.105	CNYCurrency Class Reference	264
7.106	Collar Class Reference	265
7.107	combining_iterator Class Template Reference	266
7.108	Composite Class Template Reference	268
7.109	CompositeConstraint Class Reference	269
7.110	CompositeQuote Class Template Reference	270
7.111	CompoundForward Class Reference	271
7.112	CompoundingRuleFormatter Class Reference	274
7.113	ConjugateGradient Class Reference	275

7.114	ConstantParameter Class Reference	276
7.115	Constraint Class Reference	277
7.116	ConstraintImpl Class Reference	278
7.117	ContinuousAveragingAsianOption Class Reference	279
7.118	ContinuousAveragingAsianOption::arguments Class Reference	281
7.119	ContinuousAveragingAsianOption::engine Class Reference	282
7.120	COPCurrency Class Reference	283
7.121	Copenhagen Class Reference	284
7.122	CostFunction Class Reference	285
7.123	coupling_iterator Class Template Reference	286
7.124	Coupon Class Reference	288
7.125	CovarianceDecomposition Class Reference	290
7.126	CoxIngersollRoss Class Reference	291
7.127	CoxIngersollRoss::Dynamics Class Reference	293
7.128	CoxRossRubinstein Class Reference	294
7.129	CrankNicolson Class Template Reference	295
7.130	Cubic Class Reference	296
7.131	CubicSpline Class Reference	297
7.132	CumulativeBinomialDistribution Class Reference	299
7.133	CumulativeNormalDistribution Class Reference	300
7.134	CumulativePoissonDistribution Class Reference	301
7.135	CuriouslyRecurringTemplate Class Template Reference	302
7.136	Currency Class Reference	303
7.137	CurrencyFormatter Class Reference	307
7.138	CYPCurrency Class Reference	308
7.139	CZKCurrency Class Reference	309
7.140	Date Class Reference	310
7.141	DateFormatter Class Reference	315
7.142	DayCounter Class Reference	316
7.143	DayCounterImpl Class Reference	318
7.144	DecimalFormatter Class Reference	319
7.145	DEMCurrency Class Reference	320
7.146	DepositRateHelper Class Reference	321
7.147	DerivedQuote Class Template Reference	323
7.148	DirichletBC Class Reference	324
7.149	DiscountCurve Class Reference	325

7.150	DiscountStructure Class Reference	327
7.151	DiscrepancyStatistics Class Reference	329
7.152	DiscreteAveragingAsianOption Class Reference	330
7.153	DiscreteAveragingAsianOption::arguments Class Reference	332
7.154	DiscreteAveragingAsianOption::engine Class Reference	333
7.155	DiscreteGeometricAPO Class Reference	334
7.156	DiscreteGeometricASO Class Reference	335
7.157	DiscretizedAsset Class Reference	336
7.158	DiscretizedDiscountBond Class Reference	339
7.159	DiscretizedOption Class Reference	340
7.160	Disposable Class Template Reference	342
7.161	DividendVanillaOption Class Reference	343
7.162	DividendVanillaOption::arguments Class Reference	345
7.163	DividendVanillaOption::engine Class Reference	346
7.164	DKKCurrency Class Reference	347
7.165	DMinus Class Reference	348
7.166	DownRounding Class Reference	349
7.167	DPlus Class Reference	350
7.168	DPlusDMinus Class Reference	351
7.169	DriftTermStructure Class Reference	352
7.170	DZero Class Reference	354
7.171	EarlyExercise Class Reference	355
7.172	EEKCurrency Class Reference	356
7.173	EndCriteria Class Reference	357
7.174	EqualJumpsBinomialTree Class Reference	359
7.175	EqualProbabilitiesBinomialTree Class Reference	360
7.176	Error Class Reference	361
7.177	ErrorFunction Class Reference	362
7.178	ESPCurrency Class Reference	363
7.179	EulerDiscretization Class Reference	364
7.180	EURCurrency Class Reference	365
7.181	Euribor Class Reference	366
7.182	EuroFormatter Class Reference	367
7.183	EuropeanExercise Class Reference	368
7.184	EuropeanOption Class Reference	369
7.185	ExchangeRate Class Reference	370

7.186	ExchangeRateManager Class Reference	372
7.187	Exercise Class Reference	374
7.188	ExplicitEuler Class Template Reference	375
7.189	ExtendedCoxIngersollRoss Class Reference	376
7.190	ExtendedCoxIngersollRoss::Dynamics Class Reference	377
7.191	ExtendedCoxIngersollRoss::FittingParameter Class Reference	378
7.192	ExtendedDiscountCurve Class Reference	379
7.193	Extrapolator Class Reference	381
7.194	Factorial Class Reference	382
7.195	FalsePosition Class Reference	383
7.196	FaureRsg Class Reference	384
7.197	FdAmericanOption Class Reference	385
7.198	FdBermudanOption Class Reference	386
7.199	FdBsmOption Class Reference	387
7.200	FdDividendAmericanOption Class Reference	389
7.201	FdDividendShoutOption Class Reference	390
7.202	FdEuropean Class Reference	391
7.203	FdStepConditionOption Class Reference	392
7.204	filtering_iterator Class Template Reference	393
7.205	FIMCurrency Class Reference	394
7.206	FiniteDifferenceModel Class Template Reference	395
7.207	FixedCouponBond Class Reference	396
7.208	FixedRateCoupon Class Reference	397
7.209	FlatForward Class Reference	399
7.210	FloatingRateCoupon Class Reference	401
7.211	Floor Class Reference	403
7.212	FloorTruncation Class Reference	404
7.213	ForwardEngine Class Template Reference	405
7.214	ForwardOptionArguments Class Template Reference	406
7.215	ForwardPerformanceEngine Class Template Reference	407
7.216	ForwardRateStructure Class Reference	408
7.217	ForwardSpreadedTermStructure Class Reference	410
7.218	ForwardVanillaOption Class Reference	412
7.219	FraRateHelper Class Reference	414
7.220	FrequencyFormatter Class Reference	416
7.221	FRFCurrency Class Reference	417

7.222	FuturesRateHelper Class Reference	418
7.223	G2 Class Reference	419
7.224	G2::FittingParameter Class Reference	421
7.225	G2SwaptionEngine Class Reference	422
7.226	GammaFunction Class Reference	423
7.227	GapPayoff Class Reference	424
7.228	GaussianStatistics Class Template Reference	425
7.229	GBPCurrency Class Reference	428
7.230	GBPLibor Class Reference	429
7.231	GeneralStatistics Class Reference	430
7.232	GenericEngine Class Template Reference	433
7.233	GenericModelEngine Class Template Reference	434
7.234	GenericRiskStatistics Class Template Reference	435
7.235	GeometricBrownianMotionProcess Class Reference	438
7.236	Germany Class Reference	439
7.237	GRDCurrency Class Reference	442
7.238	Greeks Class Reference	443
7.239	HaltonRsg Class Reference	444
7.240	Handle Class Template Reference	445
7.241	Helsinki Class Reference	447
7.242	History Class Reference	448
7.243	History::const_iterator Class Reference	451
7.244	History::Entry Class Reference	452
7.245	HKDCurrency Class Reference	453
7.246	HongKong Class Reference	454
7.247	HUFCurrency Class Reference	455
7.248	HullWhite Class Reference	456
7.249	HullWhite::Dynamics Class Reference	457
7.250	HullWhite::FittingParameter Class Reference	458
7.251	ICGaussianRng Class Template Reference	459
7.252	ICGaussianRsg Class Template Reference	460
7.253	IEPCurrency Class Reference	461
7.254	ILSCurrency Class Reference	462
7.255	ImplicitEuler Class Template Reference	463
7.256	ImpliedTermStructure Class Reference	464
7.257	ImpliedVolTermStructure Class Reference	466

7.258	InArrearIndexedCoupon Class Reference	468
7.259	IncrementalStatistics Class Reference	470
7.260	Index Class Reference	473
7.261	IndexedCoupon Class Reference	474
7.262	IndexManager Class Reference	476
7.263	INRCurrency Class Reference	477
7.264	Instrument Class Reference	478
7.265	IntegerFormatter Class Reference	481
7.266	IntegralEngine Class Reference	482
7.267	InterestRate Class Reference	483
7.268	InterestRateFormatter Class Reference	486
7.269	Interpolation Class Reference	487
7.270	Interpolation2D Class Reference	488
7.271	Interpolation2D::templateImpl Class Template Reference	489
7.272	Interpolation2DImpl Class Reference	490
7.273	Interpolation::templateImpl Class Template Reference	491
7.274	InterpolationImpl Class Reference	492
7.275	InverseCumulativeNormal Class Reference	493
7.276	InverseCumulativePoisson Class Reference	494
7.277	InverseCumulativeRng Class Template Reference	495
7.278	InverseCumulativeRsg Class Template Reference	496
7.279	IQDCurrency Class Reference	497
7.280	IRRCurrency Class Reference	498
7.281	ISKCurrency Class Reference	499
7.282	Italy Class Reference	500
7.283	ITLCurrency Class Reference	502
7.284	JamshidianSwaptionEngine Class Reference	503
7.285	JarrowRudd Class Reference	504
7.286	Johannesburg Class Reference	505
7.287	JointCalendar Class Reference	506
7.288	JPYCurrency Class Reference	507
7.289	JPYLibor Class Reference	508
7.290	JumpDiffusionEngine Class Reference	509
7.291	JuQuadraticApproximationEngine Class Reference	510
7.292	KnuthUniformRng Class Reference	511
7.293	KronrodIntegral Class Reference	512

7.294	KRWCurrency Class Reference	513
7.295	KWDCurrency Class Reference	514
7.296	Lattice Class Reference	515
7.297	Lattice2D Class Reference	517
7.298	LatticeShortRateModelEngine Class Template Reference	518
7.299	LazyObject Class Reference	519
7.300	LeastSquareFunction Class Reference	521
7.301	LeastSquareProblem Class Reference	522
7.302	LecuyerUniformRng Class Reference	523
7.303	LeisenReimer Class Reference	524
7.304	LexicographicalView Class Template Reference	525
7.305	Linear Class Reference	527
7.306	LinearInterpolation Class Reference	528
7.307	LineSearch Class Reference	529
7.308	Link Class Template Reference	531
7.309	LocalConstantVol Class Reference	533
7.310	LocalVolCurve Class Reference	534
7.311	LocalVolSurface Class Reference	536
7.312	LocalVolTermStructure Class Reference	538
7.313	LogLinear Class Reference	540
7.314	LogLinearInterpolation Class Reference	541
7.315	lowest_category_iterator Struct Template Reference	542
7.316	LTLCurrency Class Reference	543
7.317	LUFCurrency Class Reference	544
7.318	LVLCurrency Class Reference	545
7.319	MakeSchedule Class Reference	546
7.320	Matrix Class Reference	547
7.321	MatrixFormatter Class Reference	551
7.322	MCAmericanBasketEngine Class Reference	552
7.323	MCBarrierEngine Class Template Reference	553
7.324	MCBasketEngine Class Template Reference	555
7.325	McCliquetOption Class Reference	557
7.326	MCDigitalEngine Class Template Reference	558
7.327	MCDiscreteArithmeticAPEngine Class Template Reference	560
7.328	McDiscreteArithmeticAPO Class Reference	562
7.329	McDiscreteArithmeticASO Class Reference	563

7.330	MCDiscreteAveragingAsianEngine Class Template Reference	564
7.331	MCDiscreteGeometricAPEngine Class Template Reference	566
7.332	MCEuropeanEngine Class Template Reference	567
7.333	McEverest Class Reference	568
7.334	McHimalaya Class Reference	569
7.335	McMaxBasket Class Reference	570
7.336	McPagoda Class Reference	571
7.337	McPerformanceOption Class Reference	572
7.338	McPricer Class Template Reference	573
7.339	McSimulation Class Template Reference	574
7.340	MCVanillaEngine Class Template Reference	576
7.341	MersenneTwisterUniformRng Class Reference	578
7.342	Merton76Process Class Reference	579
7.343	MixedScheme Class Template Reference	581
7.344	Money Class Reference	583
7.345	MoneyFormatter Class Reference	585
7.346	MonotonicCubicSpline Class Reference	586
7.347	MonteCarloModel Class Template Reference	587
7.348	MoreGreeks Class Reference	588
7.349	MoroInverseCumulativeNormal Class Reference	589
7.350	MTLCurrency Class Reference	590
7.351	MultiAssetOption Class Reference	591
7.352	MultiAssetOption::arguments Class Reference	593
7.353	MultiAssetOption::results Class Reference	594
7.354	MultiCubicSpline Class Template Reference	595
7.355	MultiPath Class Reference	596
7.356	MultiPathGenerator Class Template Reference	597
7.357	MXNCurrency Class Reference	598
7.358	NaturalCubicSpline Class Reference	599
7.359	NaturalMonotonicCubicSpline Class Reference	600
7.360	NeumannBC Class Reference	601
7.361	Newton Class Reference	602
7.362	NewtonSafe Class Reference	603
7.363	NLGCurrency Class Reference	604
7.364	NoConstraint Class Reference	605
7.365	NOKCurrency Class Reference	606

7.366	NonLinearLeastSquare Class Reference	607
7.367	NormalDistribution Class Reference	608
7.368	NPRCurrency Class Reference	609
7.369	Null Class Template Reference	610
7.370	NullCalendar Class Reference	611
7.371	NullParameter Class Reference	612
7.372	NumericalMethod Class Reference	613
7.373	NZDCurrency Class Reference	615
7.374	Observable Class Reference	616
7.375	Observer Class Reference	617
7.376	OneAssetOption Class Reference	619
7.377	OneAssetOption::arguments Class Reference	622
7.378	OneAssetOption::results Class Reference	623
7.379	OneAssetStrikedOption Class Reference	624
7.380	OneDayCounter Class Reference	626
7.381	OneFactorAffineModel Class Reference	627
7.382	OneFactorModel Class Reference	628
7.383	OneFactorModel::ShortRateDynamics Class Reference	629
7.384	OneFactorModel::ShortRateTree Class Reference	630
7.385	OneFactorOperator Class Reference	631
7.386	OptimizationMethod Class Reference	632
7.387	Option Class Reference	634
7.388	Option::arguments Class Reference	635
7.389	OptionTypeFormatter Class Reference	636
7.390	OrnsteinUhlenbeckProcess Class Reference	637
7.391	Oslo Class Reference	639
7.392	Parameter Class Reference	640
7.393	ParameterImpl Class Reference	641
7.394	ParCoupon Class Reference	642
7.395	Path Class Reference	644
7.396	PathGenerator Class Template Reference	645
7.397	PathPricer Class Template Reference	646
7.398	Payoff Class Reference	647
7.399	PercentageStrikePayoff Class Reference	648
7.400	Period Class Reference	649
7.401	PiecewiseConstantParameter Class Reference	650

7.402	PiecewiseFlatForward Class Reference	651
7.403	PKRCurrency Class Reference	654
7.404	PlainVanillaPayoff Class Reference	655
7.405	PLNCurrency Class Reference	656
7.406	PoissonDistribution Class Reference	657
7.407	PositiveConstraint Class Reference	658
7.408	PricingEngine Class Reference	659
7.409	PrimeNumbers Class Reference	660
7.410	Problem Class Reference	661
7.411	processing_iterator Class Template Reference	662
7.412	PTECurrency Class Reference	664
7.413	QuantoEngine Class Template Reference	665
7.414	QuantoForwardVanillaOption Class Reference	667
7.415	QuantoOptionArguments Class Template Reference	669
7.416	QuantoOptionResults Class Template Reference	670
7.417	QuantoTermStructure Class Reference	671
7.418	QuantoVanillaOption Class Reference	673
7.419	Quote Class Reference	675
7.420	RamdomizedLDS Class Template Reference	676
7.421	RandomSequenceGenerator Class Template Reference	678
7.422	RateFormatter Class Reference	679
7.423	RateHelper Class Reference	680
7.424	Results Class Reference	682
7.425	Ridder Class Reference	683
7.426	Riyadh Class Reference	684
7.427	ROLCurrency Class Reference	685
7.428	Rounding Class Reference	686
7.429	SalvagingAlgorithm Struct Reference	688
7.430	Sample Struct Template Reference	689
7.431	SARCurrency Class Reference	690
7.432	Schedule Class Reference	691
7.433	Secant Class Reference	692
7.434	SeedGenerator Class Reference	693
7.435	SegmentIntegral Class Reference	694
7.436	SEKCurrency Class Reference	695
7.437	Seoul Class Reference	696

7.438	SequenceFormatter Class Reference	697
7.439	SequenceStatistics Class Template Reference	698
7.440	Settings Class Reference	700
7.441	SGDCurrency Class Reference	702
7.442	Short Class Template Reference	703
7.443	Short< ParCoupon > Class Template Reference	704
7.444	ShortRateModel Class Reference	705
7.445	ShoutCondition Class Reference	707
7.446	SimpleCashFlow Class Reference	708
7.447	SimpleDayCounter Class Reference	709
7.448	SimpleQuote Class Reference	710
7.449	SimpleSwap Class Reference	711
7.450	SimpleSwap::arguments Class Reference	713
7.451	SimpleSwap::results Class Reference	714
7.452	Simplex Class Reference	715
7.453	SimpsonIntegral Class Reference	716
7.454	Singapore Class Reference	717
7.455	SingleAssetOption Class Reference	718
7.456	Singleton Class Template Reference	720
7.457	SITCurrency Class Reference	721
7.458	SizeFormatter Class Reference	722
7.459	SKKCurrency Class Reference	723
7.460	SobolRsg Class Reference	724
7.461	Solver1D Class Template Reference	726
7.462	SquareRootProcess Class Reference	728
7.463	StatsHolder Class Reference	729
7.464	SteepestDescent Class Reference	730
7.465	step_iterator Class Template Reference	731
7.466	StepCondition Class Template Reference	732
7.467	stepping_iterator Class Template Reference	733
7.468	StochasticProcess Class Reference	735
7.469	StochasticProcess::discretization Class Reference	737
7.470	Stock Class Reference	738
7.471	Stockholm Class Reference	739
7.472	StrikedTypePayoff Class Reference	740
7.473	StringFormatter Class Reference	741

7.474	StulzEngine Class Reference	742
7.475	SuperSharePayoff Class Reference	743
7.476	SVD Class Reference	744
7.477	Swap Class Reference	745
7.478	SwapRateHelper Class Reference	747
7.479	Swaption Class Reference	749
7.480	Swaption::arguments Class Reference	750
7.481	Swaption::results Class Reference	751
7.482	SwaptionVolatilityMatrix Class Reference	752
7.483	SwaptionVolatilityStructure Class Reference	753
7.484	Sydney Class Reference	755
7.485	SymmetricSchurDecomposition Class Reference	756
7.486	Taiwan Class Reference	757
7.487	TARGET Class Reference	758
7.488	TermStructureConsistentModel Class Reference	759
7.489	TermStructureFittingParameter Class Reference	760
7.490	THBCurrency Class Reference	761
7.491	Thirty360 Class Reference	762
7.492	Tian Class Reference	763
7.493	TimeBasket Class Reference	764
7.494	TimeGrid Class Reference	765
7.495	Tokyo Class Reference	766
7.496	Toronto Class Reference	768
7.497	TrapezoidIntegral Class Reference	769
7.498	Tree Class Reference	771
7.499	TreeCapFloorEngine Class Reference	772
7.500	TreeSwaptionEngine Class Reference	773
7.501	TridiagonalOperator Class Reference	774
7.502	TridiagonalOperator::TimeSetter Class Reference	776
7.503	Trigeorgis Class Reference	777
7.504	TrinomialBranching Class Reference	778
7.505	TrinomialTree Class Reference	779
7.506	TRLCurrency Class Reference	780
7.507	TTDCurrency Class Reference	781
7.508	TWDCurrency Class Reference	782
7.509	TwoFactorModel Class Reference	783

7.510	TwoFactorModel::ShortRateDynamics Class Reference	784
7.511	TwoFactorModel::ShortRateTree Class Reference	785
7.512	TypePayoff Class Reference	786
7.513	UnitedKingdom Class Reference	787
7.514	UnitedStates Class Reference	789
7.515	UpFrontIndexedCoupon Class Reference	792
7.516	UpRounding Class Reference	793
7.517	USDCurrency Class Reference	794
7.518	USDLibor Class Reference	795
7.519	Value Class Reference	796
7.520	VanillaOption Class Reference	797
7.521	VanillaOption::engine Class Reference	798
7.522	Vasicek Class Reference	799
7.523	Vasicek::Dynamics Class Reference	800
7.524	VEBCurrency Class Reference	801
7.525	Visitor Class Template Reference	802
7.526	VolatilityFormatter Class Reference	803
7.527	Warsaw Class Reference	804
7.528	WeekdayFormatter Class Reference	805
7.529	Wellington Class Reference	806
7.530	Xibor Class Reference	807
7.531	XiborManager Class Reference	809
7.532	YieldTermStructure Class Reference	810
7.533	ZARCurrency Class Reference	817
7.534	ZARLibor Class Reference	818
7.535	ZeroCurve Class Reference	819
7.536	ZeroSpreadedTermStructure Class Reference	821
7.537	ZeroYieldStructure Class Reference	823
7.538	Zurich Class Reference	825
8	QuantLib File Documentation	827
8.1	ql/argsandresults.hpp File Reference	827
8.2	ql/basetermstructure.hpp File Reference	829
8.3	ql/basicdataformatters.hpp File Reference	830
8.4	ql/calendar.hpp File Reference	831
8.5	ql/Calendars/beijing.hpp File Reference	832
8.6	ql/Calendars/budapest.hpp File Reference	833

8.7	ql/Calendars/copenhagen.hpp File Reference	834
8.8	ql/Calendars/germany.hpp File Reference	835
8.9	ql/Calendars/helsinki.hpp File Reference	836
8.10	ql/Calendars/hongkong.hpp File Reference	837
8.11	ql/Calendars/italy.hpp File Reference	838
8.12	ql/Calendars/johannesburg.hpp File Reference	839
8.13	ql/Calendars/jointcalendar.hpp File Reference	840
8.14	ql/Calendars/nullcalendar.hpp File Reference	841
8.15	ql/Calendars/oslo.hpp File Reference	842
8.16	ql/Calendars/riyadh.hpp File Reference	843
8.17	ql/Calendars/seoul.hpp File Reference	844
8.18	ql/Calendars/singapore.hpp File Reference	845
8.19	ql/Calendars/stockholm.hpp File Reference	846
8.20	ql/Calendars/sydney.hpp File Reference	847
8.21	ql/Calendars/taiwan.hpp File Reference	848
8.22	ql/Calendars/target.hpp File Reference	849
8.23	ql/Calendars/tokyo.hpp File Reference	850
8.24	ql/Calendars/toronto.hpp File Reference	851
8.25	ql/Calendars/unitedkingdom.hpp File Reference	852
8.26	ql/Calendars/unitedstates.hpp File Reference	853
8.27	ql/Calendars/warsaw.hpp File Reference	854
8.28	ql/Calendars/wellington.hpp File Reference	855
8.29	ql/Calendars/zurich.hpp File Reference	856
8.30	ql/capvolstructures.hpp File Reference	857
8.31	ql/cashflow.hpp File Reference	859
8.32	ql/CashFlows/basispointsensitivity.hpp File Reference	860
8.33	ql/CashFlows/cashflowvectors.hpp File Reference	861
8.34	ql/CashFlows/coupon.hpp File Reference	863
8.35	ql/CashFlows/fixedratecoupon.hpp File Reference	864
8.36	ql/CashFlows/floatingratecoupon.hpp File Reference	865
8.37	ql/CashFlows/inarrearindexedcoupon.hpp File Reference	866
8.38	ql/CashFlows/indexcashflowvectors.hpp File Reference	867
8.39	ql/CashFlows/indexedcoupon.hpp File Reference	869
8.40	ql/CashFlows/parcoupon.hpp File Reference	870
8.41	ql/CashFlows/shortfloatingcoupon.hpp File Reference	871
8.42	ql/CashFlows/shortindexedcoupon.hpp File Reference	872

8.43	ql/CashFlows/simplecashflow.hpp File Reference	873
8.44	ql/CashFlows/timebasket.hpp File Reference	874
8.45	ql/CashFlows/upfrontindexedcoupon.hpp File Reference	875
8.46	ql/Currencies/africa.hpp File Reference	876
8.47	ql/Currencies/america.hpp File Reference	877
8.48	ql/Currencies/asia.hpp File Reference	879
8.49	ql/Currencies/europe.hpp File Reference	881
8.50	ql/Currencies/exchangeratemanager.hpp File Reference	884
8.51	ql/Currencies/oceania.hpp File Reference	885
8.52	ql/currency.hpp File Reference	886
8.53	ql/dataformatters.hpp File Reference	888
8.54	ql/dataparsers.hpp File Reference	889
8.55	ql/date.hpp File Reference	890
8.56	ql/daycounter.hpp File Reference	892
8.57	ql/DayCounters/actual360.hpp File Reference	893
8.58	ql/DayCounters/actual365.hpp File Reference	894
8.59	ql/DayCounters/actual365fixed.hpp File Reference	895
8.60	ql/DayCounters/actualactual.hpp File Reference	896
8.61	ql/DayCounters/one.hpp File Reference	897
8.62	ql/DayCounters/simpliedaycounter.hpp File Reference	898
8.63	ql/DayCounters/thirty360.hpp File Reference	899
8.64	ql/discretizedasset.hpp File Reference	900
8.65	ql/disposable.hpp File Reference	901
8.66	ql/errors.hpp File Reference	902
8.67	ql/exchangerate.hpp File Reference	904
8.68	ql/exercise.hpp File Reference	905
8.69	ql/FiniteDifferences/americancondition.hpp File Reference	906
8.70	ql/FiniteDifferences/boundarycondition.hpp File Reference	907
8.71	ql/FiniteDifferences/bsmoperator.hpp File Reference	908
8.72	ql/FiniteDifferences/cranknicolson.hpp File Reference	909
8.73	ql/FiniteDifferences/dminus.hpp File Reference	910
8.74	ql/FiniteDifferences/dplus.hpp File Reference	911
8.75	ql/FiniteDifferences/dplusdminus.hpp File Reference	912
8.76	ql/FiniteDifferences/dzero.hpp File Reference	913
8.77	ql/FiniteDifferences/expliciteuler.hpp File Reference	914
8.78	ql/FiniteDifferences/fdtypedefs.hpp File Reference	915

8.79	ql/FiniteDifferences/finitedifferencemodel.hpp File Reference	916
8.80	ql/FiniteDifferences/impliciteuler.hpp File Reference	917
8.81	ql/FiniteDifferences/mixedscheme.hpp File Reference	918
8.82	ql/FiniteDifferences/onefactoroperator.hpp File Reference	919
8.83	ql/FiniteDifferences/shoutcondition.hpp File Reference	920
8.84	ql/FiniteDifferences/stepcondition.hpp File Reference	921
8.85	ql/FiniteDifferences/tridiagonaloperator.hpp File Reference	922
8.86	ql/FiniteDifferences/valueatcenter.hpp File Reference	923
8.87	ql/grid.hpp File Reference	925
8.88	ql/history.hpp File Reference	926
8.89	ql/index.hpp File Reference	927
8.90	ql/Indexes/audlibor.hpp File Reference	928
8.91	ql/Indexes/cadlibor.hpp File Reference	929
8.92	ql/Indexes/chflibor.hpp File Reference	930
8.93	ql/Indexes/euribor.hpp File Reference	931
8.94	ql/Indexes/gbplibor.hpp File Reference	932
8.95	ql/Indexes/indexmanager.hpp File Reference	933
8.96	ql/Indexes/jpylibor.hpp File Reference	934
8.97	ql/Indexes/usdlibor.hpp File Reference	935
8.98	ql/Indexes/xibor.hpp File Reference	936
8.99	ql/Indexes/xibormanager.hpp File Reference	937
8.100	ql/Indexes/zarlibor.hpp File Reference	938
8.101	ql/instrument.hpp File Reference	939
8.102	ql/Instruments/asianoption.hpp File Reference	940
8.103	ql/Instruments/barrieroption.hpp File Reference	941
8.104	ql/Instruments/basketoption.hpp File Reference	942
8.105	ql/Instruments/bond.hpp File Reference	943
8.106	ql/Instruments/capfloor.hpp File Reference	944
8.107	ql/Instruments/cliquetoption.hpp File Reference	945
8.108	ql/Instruments/dividendvanillaoption.hpp File Reference	946
8.109	ql/Instruments/europeanoption.hpp File Reference	947
8.110	ql/Instruments/fixedcouponbond.hpp File Reference	948
8.111	ql/Instruments/forwardvanillaoption.hpp File Reference	949
8.112	ql/Instruments/multiassetoption.hpp File Reference	950
8.113	ql/Instruments/oneassetoption.hpp File Reference	951
8.114	ql/Instruments/oneassetstrikedoption.hpp File Reference	952

8.115	ql/Instruments/payoffs.hpp File Reference	953
8.116	ql/Instruments/quantoforwardvanillaoption.hpp File Reference	955
8.117	ql/Instruments/quantovanillaoption.hpp File Reference	956
8.118	ql/Instruments/simpleswap.hpp File Reference	957
8.119	ql/Instruments/stock.hpp File Reference	958
8.120	ql/Instruments/swap.hpp File Reference	959
8.121	ql/Instruments/swaption.hpp File Reference	960
8.122	ql/Instruments/vanillaoption.hpp File Reference	961
8.123	ql/interestrates.hpp File Reference	962
8.124	ql/Lattices/binomialtree.hpp File Reference	963
8.125	ql/Lattices/bsmllattice.hpp File Reference	965
8.126	ql/Lattices/lattice.hpp File Reference	966
8.127	ql/Lattices/lattice2d.hpp File Reference	967
8.128	ql/Lattices/tree.hpp File Reference	968
8.129	ql/Lattices/trinomialtree.hpp File Reference	969
8.130	ql/Math/array.hpp File Reference	970
8.131	ql/Math/beta.hpp File Reference	971
8.132	ql/Math/bicubicsplineinterpolation.hpp File Reference	972
8.133	ql/Math/bilinearinterpolation.hpp File Reference	973
8.134	ql/Math/binomialdistribution.hpp File Reference	974
8.135	ql/Math/bivariatenormaldistribution.hpp File Reference	976
8.136	ql/Math/chisquaredistribution.hpp File Reference	977
8.137	ql/Math/choleskydecomposition.hpp File Reference	978
8.138	ql/Math/comparison.hpp File Reference	979
8.139	ql/Math/cubicspline.hpp File Reference	980
8.140	ql/Math/discrepancystatistics.hpp File Reference	981
8.141	ql/Math/errorfunction.hpp File Reference	982
8.142	ql/Math/extrapolation.hpp File Reference	983
8.143	ql/Math/factorial.hpp File Reference	984
8.144	ql/Math/functional.hpp File Reference	985
8.145	ql/Math/gammadistribution.hpp File Reference	986
8.146	ql/Math/gaussianstatistics.hpp File Reference	987
8.147	ql/Math/generalstatistics.hpp File Reference	988
8.148	ql/Math/incompletegammagamma.hpp File Reference	989
8.149	ql/Math/incrementalstatistics.hpp File Reference	990
8.150	ql/Math/interpolation.hpp File Reference	991

8.151	ql/Math/interpolation2D.hpp File Reference	992
8.152	ql/Math/interpolationtraits.hpp File Reference	993
8.153	ql/Math/kronrodintegral.hpp File Reference	994
8.154	ql/Math/lexicographicalview.hpp File Reference	995
8.155	ql/Math/linearinterpolation.hpp File Reference	996
8.156	ql/Math/loglinearinterpolation.hpp File Reference	997
8.157	ql/Math/matrix.hpp File Reference	998
8.158	ql/Math/multicubicspline.hpp File Reference	999
8.159	ql/Math/normaldistribution.hpp File Reference	1001
8.160	ql/Math/poissondistribution.hpp File Reference	1002
8.161	ql/Math/primenumbers.hpp File Reference	1003
8.162	ql/Math/pseudosqrt.hpp File Reference	1004
8.163	ql/Math/riskstatistics.hpp File Reference	1005
8.164	ql/Math/rounding.hpp File Reference	1007
8.165	ql/Math/segmentintegral.hpp File Reference	1008
8.166	ql/Math/sequencestatistics.hpp File Reference	1009
8.167	ql/Math/simpsonintegral.hpp File Reference	1011
8.168	ql/Math/statistics.hpp File Reference	1012
8.169	ql/Math/svd.hpp File Reference	1013
8.170	ql/Math/symmetriceigenvalues.hpp File Reference	1014
8.171	ql/Math/symmetricschurdecomposition.hpp File Reference	1015
8.172	ql/Math/trapezoidintegral.hpp File Reference	1016
8.173	ql/money.hpp File Reference	1017
8.174	ql/MonteCarlo/brownianbridge.hpp File Reference	1018
8.175	ql/MonteCarlo/getcovariance.hpp File Reference	1019
8.176	ql/MonteCarlo/mctraits.hpp File Reference	1020
8.177	ql/MonteCarlo/mctypedefs.hpp File Reference	1021
8.178	ql/MonteCarlo/montecarlomodel.hpp File Reference	1022
8.179	ql/MonteCarlo/multipath.hpp File Reference	1023
8.180	ql/MonteCarlo/multipathgenerator.hpp File Reference	1024
8.181	ql/MonteCarlo/path.hpp File Reference	1025
8.182	ql/MonteCarlo/pathgenerator.hpp File Reference	1026
8.183	ql/MonteCarlo/pathpricer.hpp File Reference	1027
8.184	ql/MonteCarlo/sample.hpp File Reference	1028
8.185	ql/null.hpp File Reference	1029
8.186	ql/numericalmethod.hpp File Reference	1030

8.187	ql/Optimization/armijo.hpp File Reference	1031
8.188	ql/Optimization/conjugategradient.hpp File Reference	1032
8.189	ql/Optimization/constraint.hpp File Reference	1033
8.190	ql/Optimization/costfunction.hpp File Reference	1034
8.191	ql/Optimization/criteria.hpp File Reference	1035
8.192	ql/Optimization/leastsquare.hpp File Reference	1036
8.193	ql/Optimization/linesearch.hpp File Reference	1037
8.194	ql/Optimization/method.hpp File Reference	1038
8.195	ql/Optimization/problem.hpp File Reference	1039
8.196	ql/Optimization/simplex.hpp File Reference	1040
8.197	ql/Optimization/steepestdescent.hpp File Reference	1041
8.198	ql/option.hpp File Reference	1042
8.199	ql/Patterns/bridge.hpp File Reference	1043
8.200	ql/Patterns/composite.hpp File Reference	1044
8.201	ql/Patterns/curiouslyrecurring.hpp File Reference	1045
8.202	ql/Patterns/lazyobject.hpp File Reference	1046
8.203	ql/Patterns/observable.hpp File Reference	1047
8.204	ql/Patterns/singleton.hpp File Reference	1048
8.205	ql/Patterns/visitor.hpp File Reference	1049
8.206	ql/payoff.hpp File Reference	1050
8.207	ql/Pricers/discretegeometricapo.hpp File Reference	1051
8.208	ql/Pricers/discretegeometricaso.hpp File Reference	1052
8.209	ql/Pricers/fdamericanoption.hpp File Reference	1053
8.210	ql/Pricers/fdbermudanoption.hpp File Reference	1054
8.211	ql/Pricers/fdbsmoption.hpp File Reference	1055
8.212	ql/Pricers/fddividendamERICANoption.hpp File Reference	1056
8.213	ql/Pricers/fddividendoption.hpp File Reference	1057
8.214	ql/Pricers/fddividendshoutoption.hpp File Reference	1058
8.215	ql/Pricers/fdeuropean.hpp File Reference	1059
8.216	ql/Pricers/fdmultiPERIODoption.hpp File Reference	1060
8.217	ql/Pricers/fdshoutoption.hpp File Reference	1061
8.218	ql/Pricers/fdstepconditionoption.hpp File Reference	1062
8.219	ql/Pricers/mccliQUEToption.hpp File Reference	1063
8.220	ql/Pricers/mcdiscretearithMETICapo.hpp File Reference	1064
8.221	ql/Pricers/mcdiscretearithMETICaso.hpp File Reference	1065
8.222	ql/Pricers/mceverest.hpp File Reference	1066

8.223	ql/Pricers/mchimalaya.hpp File Reference	1067
8.224	ql/Pricers/mcmaxbasket.hpp File Reference	1068
8.225	ql/Pricers/mcpagoda.hpp File Reference	1069
8.226	ql/Pricers/mcperformanceoption.hpp File Reference	1070
8.227	ql/Pricers/mcpricer.hpp File Reference	1071
8.228	ql/Pricers/singleassetoption.hpp File Reference	1072
8.229	ql/pricingengine.hpp File Reference	1073
8.230	ql/PricingEngines/americanpayoffatexpiry.hpp File Reference	1074
8.231	ql/PricingEngines/americanpayoffathit.hpp File Reference	1075
8.232	ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp File Reference	1076
8.233	ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp File Reference	1077
8.234	ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp File Reference	1078
8.235	ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp File Reference	1079
8.236	ql/PricingEngines/Asian/mcdiscreteasianengine.hpp File Reference	1080
8.237	ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference	1081
8.238	ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference	1082
8.239	ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference	1083
8.240	ql/PricingEngines/Basket/mcbasketengine.hpp File Reference	1084
8.241	ql/PricingEngines/Basket/stulzengine.hpp File Reference	1085
8.242	ql/PricingEngines/blackformula.hpp File Reference	1086
8.243	ql/PricingEngines/blackmodel.hpp File Reference	1087
8.244	ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference	1088
8.245	ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference	1089
8.246	ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference	1090
8.247	ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference	1091
8.248	ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference	1092
8.249	ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference	1093
8.250	ql/PricingEngines/Cliquet/mccliquetengine.hpp File Reference	1094
8.251	ql/PricingEngines/Forward/forwardengine.hpp File Reference	1095
8.252	ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference	1096
8.253	ql/PricingEngines/genericmodelengine.hpp File Reference	1097
8.254	ql/PricingEngines/latticeshortratemodelengine.hpp File Reference	1098
8.255	ql/PricingEngines/mcsimulation.hpp File Reference	1099
8.256	ql/PricingEngines/Quanto/quantoengine.hpp File Reference	1100
8.257	ql/PricingEngines/Swapoption/blackswapoptionengine.hpp File Reference	1101
8.258	ql/PricingEngines/Swapoption/discretizedswapoption.hpp File Reference	1102

8.259	ql/PricingEngines/Swaption/g2swaptionengine.hpp File Reference	1103
8.260	ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp File Reference . . .	1104
8.261	ql/PricingEngines/Swaption/treeswaptionengine.hpp File Reference	1105
8.262	ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference . .	1106
8.263	ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp File Reference .	1107
8.264	ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference	1108
8.265	ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference	1109
8.266	ql/PricingEngines/Vanilla/binomialengine.hpp File Reference	1110
8.267	ql/PricingEngines/Vanilla/bjerk sundstenslandengine.hpp File Reference	1111
8.268	ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference	1112
8.269	ql/PricingEngines/Vanilla/integralengine.hpp File Reference	1113
8.270	ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference	1114
8.271	ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference	1115
8.272	ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference	1116
8.273	ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference	1117
8.274	ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference	1118
8.275	ql/qldefines.hpp File Reference	1119
8.276	ql/quote.hpp File Reference	1121
8.277	ql/RandomNumbers/boxmullergaussianrng.hpp File Reference	1122
8.278	ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference	1123
8.279	ql/RandomNumbers/faurersg.hpp File Reference	1124
8.280	ql/RandomNumbers/haltonrng.hpp File Reference	1125
8.281	ql/RandomNumbers/inversecumgaussianrng.hpp File Reference	1126
8.282	ql/RandomNumbers/inversecumgaussianrng.hpp File Reference	1127
8.283	ql/RandomNumbers/inversecumulativerng.hpp File Reference	1128
8.284	ql/RandomNumbers/inversecumulativersg.hpp File Reference	1129
8.285	ql/RandomNumbers/knuthuniformrng.hpp File Reference	1130
8.286	ql/RandomNumbers/lecuyeruniformrng.hpp File Reference	1131
8.287	ql/RandomNumbers/mt19937uniformrng.hpp File Reference	1132
8.288	ql/RandomNumbers/randomizedlds.hpp File Reference	1133
8.289	ql/RandomNumbers/randomsequencegenerator.hpp File Reference	1134
8.290	ql/RandomNumbers/rngtraits.hpp File Reference	1135
8.291	ql/RandomNumbers/seedgenerator.hpp File Reference	1137
8.292	ql/RandomNumbers/sobolrng.hpp File Reference	1138
8.293	ql/relinkablehandle.hpp File Reference	1139
8.294	ql/schedule.hpp File Reference	1140

8.295	ql/settings.hpp File Reference	1141
8.296	ql/ShortRateModels/calibrationhelper.hpp File Reference	1142
8.297	ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference	1143
8.298	ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference	1144
8.299	ql/ShortRateModels/model.hpp File Reference	1145
8.300	ql/ShortRateModels/onefactormodel.hpp File Reference	1146
8.301	ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference	1147
8.302	ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference	1148
8.303	ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference	1149
8.304	ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference	1150
8.305	ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference	1151
8.306	ql/ShortRateModels/parameter.hpp File Reference	1152
8.307	ql/ShortRateModels/twofactormodel.hpp File Reference	1153
8.308	ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference	1154
8.309	ql/solver1d.hpp File Reference	1155
8.310	ql/Solvers1D/bisection.hpp File Reference	1156
8.311	ql/Solvers1D/brent.hpp File Reference	1157
8.312	ql/Solvers1D/falseposition.hpp File Reference	1158
8.313	ql/Solvers1D/newton.hpp File Reference	1159
8.314	ql/Solvers1D/newtonsafe.hpp File Reference	1160
8.315	ql/Solvers1D/ridder.hpp File Reference	1161
8.316	ql/Solvers1D/secant.hpp File Reference	1162
8.317	ql/stochasticprocess.hpp File Reference	1163
8.318	ql/swaptionvolstructure.hpp File Reference	1164
8.319	ql/termstructure.hpp File Reference	1165
8.320	ql/TermStructures/affinetermstructure.hpp File Reference	1166
8.321	ql/TermStructures/compoundforward.hpp File Reference	1167
8.322	ql/TermStructures/discountcurve.hpp File Reference	1168
8.323	ql/TermStructures/discountstructure.hpp File Reference	1169
8.324	ql/TermStructures/drifttermstructure.hpp File Reference	1170
8.325	ql/TermStructures/extendeddiscountcurve.hpp File Reference	1171
8.326	ql/TermStructures/flatforward.hpp File Reference	1172
8.327	ql/TermStructures/forwardspreadedtermstructure.hpp File Reference	1173
8.328	ql/TermStructures/forwardstructure.hpp File Reference	1174
8.329	ql/TermStructures/IMPLIEDtermstructure.hpp File Reference	1175
8.330	ql/TermStructures/piecewiseflatforward.hpp File Reference	1176

8.331	ql/TermStructures/quantotermstructure.hpp File Reference	1177
8.332	ql/TermStructures/ratehelpers.hpp File Reference	1178
8.333	ql/TermStructures/zerocurve.hpp File Reference	1179
8.334	ql/TermStructures/zerospreadedtermstructure.hpp File Reference	1180
8.335	ql/TermStructures/zeroyieldstructure.hpp File Reference	1181
8.336	ql/types.hpp File Reference	1182
8.337	ql/Utilities/combiningiterator.hpp File Reference	1184
8.338	ql/Utilities/couplingiterator.hpp File Reference	1185
8.339	ql/Utilities/filteringiterator.hpp File Reference	1186
8.340	ql/Utilities/iteratorcategories.hpp File Reference	1187
8.341	ql/Utilities/processingiterator.hpp File Reference	1188
8.342	ql/Utilities/steppingiterator.hpp File Reference	1189
8.343	ql/Volatilities/blackconstantvol.hpp File Reference	1190
8.344	ql/Volatilities/blackvariancecurve.hpp File Reference	1191
8.345	ql/Volatilities/blackvariancesurface.hpp File Reference	1192
8.346	ql/Volatilities/capflatvolvector.hpp File Reference	1193
8.347	ql/Volatilities/capletconstantvol.hpp File Reference	1194
8.348	ql/Volatilities/impliedvoltermstructure.hpp File Reference	1195
8.349	ql/Volatilities/localconstantvol.hpp File Reference	1196
8.350	ql/Volatilities/localvolcurve.hpp File Reference	1197
8.351	ql/Volatilities/localvolsurface.hpp File Reference	1198
8.352	ql/Volatilities/swaptionvolmatrix.hpp File Reference	1199
8.353	ql/voltermstructure.hpp File Reference	1200
9	QuantLib Example Documentation	1201
9.1	AmericanOption.cpp	1201
9.2	BermudanSwaption.cpp	1206
9.3	DiscreteHedging.cpp	1211
9.4	EuropeanOption.cpp	1217
9.5	history_iterators.cpp	1223
9.6	swapvaluation.cpp	1224
10	Test List	1235
11	Deprecated List	1243
12	Todo List	1249
13	Bug List	1253

Chapter 1

Getting started

1.1 Introduction

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

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

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

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

1.1.1 Disclaimer

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

1.2 Project overview

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

The following list is an overview of the existing code base.

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

Date, calendars, and day count conventions

- Date class.
- Weekday, month, frequency, time unit enumerations.
- Period class (eg. 1y, 30d, 2m, etc.)
- IMM calculation.
- Calendars: Beijing, Budapest, Copenhagen, Frankfurt, Helsinki, Hong Kong, Italy (Settlement, Exchange), Johannesburg, Oslo, Riyadh, Seoul, Singapore, Stockholm, Sydney, Taiwan, TARGET, Tokyo, Toronto, United Kingdom (Settlement, Exchange, Metals), United States (Settlement, Exchange, GovernmentBond), Xetra, Warsaw, Wellington, Zurich.
- NullCalendar (no holidays) for theoretical calculations.
- Joint calendars made up as holiday union or intersection of base calendars.
- Rolling conventions: Preceding, ModifiedPreceding, Following, ModifiedFollowing, MonthEndReference.
- Schedule class for date stream generation.
- Day count conventions: Actual360, Actual365Fixed, ActualActual (Bond, ISDA, AFB), 30/360 (US, European, Italian), 1/1.

To do:

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

Math

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

- Bivariate cumulative normal distribution.
- Binomial coefficients, binomial distribution, cumulative binomial distribution, and Peizer-Pratt inversion (method 2.)
- Chi square and non-central chi square distributions.
- Beta functions.
- Poisson and cumulative Poisson distributions.
- Incomplete gamma functions.
- Gamma distribution.
- Factorials.
- Integration algorithms: segment, trapezoid, mid-point trapezoid, Simpson, Gauss-Kronrod.
- Error function.
- General 1-D statistics: mean, variance, standard deviation, skewness, kurtosis, error estimation, min, max.
- Multi-dimensional (sequence) statistics: all the 1-D methods plus covariance, correlation, L2-discrepancy calculation, etc.
- Risk measures for Gaussian and empirical distributions: semi-variance, regret, percentile, top percentile, value-at-risk, upside potential, shortfall, average shortfall, expected shortfall.
- Array and matrix classes for algebra.
- Singular value decomposition.
- Eigenvalues, eigenvectors for symmetric matrices.
- Cholesky decomposition.
- Schur decomposition.
- Spectral rank-reduced square root, spectral pseudo-square root.

To do:

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

1-dimensional solvers

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

To do:

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

Optimization

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

Random-number generation

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

Patterns

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

Finite differences

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

To do:

- Richardson extrapolation
- Introduce variable theta schemes.
- Introduce multi time-level schemes.
- Enable different solvers (SOR, etc.)
- Extend to time-dependant parameters.
- Extend to local volatility.

- Two-dimension schemes.
- Improve boundary conditions.
- 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:

- Greeks calculation.
- Allow easier selection between Incremental/BrownianBridge path generation.

- Review Monte Carlo engine for american options.
- Review multi-factor Monte Carlo simulation.
- Predictor-corrector scheme.
- Add Milstein scheme.
- Add Martingale control variate.
- Batch samples as $N=n_batches*batch_size$, and exploit it for randomized low discrepancy sequences (RQMC)

Pricing engines

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

To do:

- Add the trigger level Touch/NoTouch specification for American-style digitals.

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

Pricers

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

To do:

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

Financial Instruments

- Instrument base class: npv(), isExpired(), etc.
- Interest-rate swap: simple, normal.
- Swaption.
- Cap/floor.

- Fixed-rate coupon bond.
- Stock.
- One-asset option base class.
- Asian option.
- Barrier option.
- Cliquet option.
- Forward vanilla option.
- Quanto vanilla option.
- Quanto-forward vanilla option.
- Vanilla option.
- Multi-asset option base class.
- Basket option.

To do:

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

Yield term structures

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

To do:

- Future convexity adjustment
- End of year effect

Volatility

- Interface for cap/floor Black volatility term structures (unstable).
- Interface for swaption Black volatility term structures (unstable).

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

To do:

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

Short rate models

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

Credit derivatives

To do:

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

Test suite

Implemented by means of the Boost unit-test framework. More than 170 automated tests. An automatically-generated list is available in [chapter 10](#).

To do:

- Add covariance/correlation test for SequenceStatistics.
- Add single factor calibration and Bermudan pricing tests.
- Increase coverage.

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 class and enumeration.
- Money class with automatic exchange-rate capabilities.
- Output data formatters: long integers, Ordinal numerals, power of two, exponential, fixed digit, sequences, dates, etc.
- Input data parsers.
- Error classes and error handling.

- Exercise classes: European, Bermudan, American
- Payoff classes: plain, gap, asset-or-nothing, cash-or-nothing
- Grid classes for handling of equally and unequally spaced grids.
- History class for handling of historical data.
- Quote class for mutable data.
- Null types.
- User-configurable #define to disable usage of deprecated classes.

To do:

- IRR, duration, convexity, etc. for a sequence of cash flows
- Implement currency as per OMG definition

Documentation

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

To do:

- Add a "Getting started" page to the site

Distribution

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

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; see the [User configuration](#) section for configuration options.
2. Type 'make' to compile the package.
3. Type 'make install' to install the library. This might require administrative privileges.

1.4.2 Win32

An installer for the source distribution is available at

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

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

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

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

If you use Borland command line compiler (5.5.1) the make options are: -D_DEBUG (debug), -D_RTLDLL (dynamic linking of runtime library), -D_MT__ (multi-thread) as in:

```
make all
```

```
make -D_DEBUG all
```

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

or any other combination of options.

1.5 User configuration

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

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

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

for a list of corresponding command-line options.

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

Such macros include:

```
#define QL_ERROR_LINES
```

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

```
#define QL_NEGATIVE_RATES
```

If defined, negative yield rates are allowed in a few places where they are currently forbidden. It is still not clear whether this is safe. The default is to undefine the macro.

```
#define QL_EXTRA_SAFETY_CHECKS
```

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

```
#define QL_TODAYS_PAYMENTS
```

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

```
#define QL_DISABLE_DEPRECATED
```

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

```
#define QL_USE_INDEXED_COUPON
```

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

1.6 Usage

To use QuantLib classes in your own code just add

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

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

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

1.6.1 Microsoft Visual C++

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

1. you won't have to explicitly link your application to the QuantLib library. This is done automatically by compiler directives embedded in the sources.
2. You have to define the NOMINMAX macro.
3. Your project must be compiled with the same options that were used in compiling the QuantLib library you're linking with. For VC6 please
 - a) select the appropriate run-time library under project settings, "C/C++" tab, "Code Generation",
 - b) check the "Use RTTI" option under the "C++ Language" category.

For VC7 (.NET) under

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

1.7 Frequently asked questions

1.7.1 Generic questions

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

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

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

```
X-Bogosity: Yes
```

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

How should I report a bug?

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

In any case please report as much details as possible.

If it is a compilation problem please state at least:

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

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

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

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

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

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

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

< broad grin > No, we aren't.

1.7.2 Contributing to the project

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

Contact either project administrators (Nando <<mailto:nando@users.sourceforge.net>> and Luigi <<mailto:lballabio@users.sourceforge.net>>) and describe your experience and interests. Before doing this, please read:

- the generic introduction for new developers <http://cvs.sourceforge.net/viewcvs.py/quantlib/QuantLib/dev_tools/newdeveloperintro.txt?view=markup>
- the project overview <<http://www.quantlib.org/reference/overview.html>>, with its to-do suggestions
- the Developer FAQ <<http://www.quantlib.org/developerFAQ.shtml>>
- The Programming Style Guidelines <<http://www.quantlib.org/style.shtml>>
- the detailed low-level to-do list <<http://www.quantlib.org/reference/todo.html>>

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

How do I contribute code to the project?

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

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

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

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

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

1.7.3 Building QuantLib

1.7.4 Using QuantLib

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

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

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

1.7.5 QuantLib features

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

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

1.8 Version history

Release 0.3.8 - December 2004

REQUIRED PACKAGES

- Boost version 1.31.0 or later is now required.

DOCUMENTATION

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

GLOBAL FEATURES

- Global evaluation date added through Settings class. Used for index-fixing and exchange-rate lookup.
- added InterestRate class, which encapsulate the interest rate compounding algebra. It manages day-counting convention, compounding convention, conversion between different conventions, and discount/compounding factor calculations. It also has its own formatter.

INSTRUMENTS

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

DATE, CALENDARS, AND DAY COUNT CONVENTIONS

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

CURRENCIES AND FX RATES

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

MONTE CARLO FRAMEWORK

- Added Faure low-discrepancy sequence (thanks to Gianni Piolanti;) tests provided.

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

LATTICE FRAMEWORK

- Reworked the DiscretizedAsset interface.

PRICING ENGINES FRAMEWORK

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

FIXED INCOME

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

YIELD TERM STRUCTURE

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

PATTERNS

- Added Singleton pattern.

MATH

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

MISCELLANEA

- Renamed RelinkableHandle to Handle.

PORTABILITY

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

Release 0.3.7 - July 23rd, 2004

IMPORTANT

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

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

DATE, CALENDARS, AND DAY COUNT CONVENTIONS

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

MATH

- Added rounding algorithms as per OMG enumeration/definition.

TEST SUITE

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

- Added test for cliquet options.

MISCELLANEA

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

PORTABILITY

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

Release 0.3.6 - April 15th, 2004

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

Release 0.3.5 - March 31th, 2004

BOOST SUPPORT

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

MONTE CARLO FRAMEWORK

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

PRICING ENGINES FRAMEWORK

- Pricing engines now use Payoff and Exercise classes.
- American basket options.
- Binary barrier option replaced by vanilla option with digital payoff.
- Stulz engine for max and min basket calls and puts on two assets.
- American binary option added (a.k.a. one-touch, american digital, american barrier, etc.) with different payoffs (cash/asset at hit/expiry, etc.)
- Added engine for Merton 1976 jump-diffusion process.

- Added Bjerk Sund and Stensland approximation for American option (still unstable.)
- Added Barone-Adesi and Whaley approximation for American option.
- Improved Black formula engine with more greeks added.
- Discrete geometric asian option.
- Added Leisen-Reimer binomial tree.

SHORT RATE MODELS

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

VOLATILITY FRAMEWORK

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

OPTIMIZATION FRAMEWORK

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

PATTERNS

- Composite pattern

MATH

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

TEST SUITE

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

MISCELLANEA

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

Release 0.3.4 - November 21th, 2003

MONTE CARLO FRAMEWORK

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

PRICING ENGINES FRAMEWORK

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

FIXED INCOME

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

SHORT RATE MODELS

- An additional constraint can now be passed to the calibration

VOLATILITY FRAMEWORK

- Visitable volatility term structures

OPTIMIZATION FRAMEWORK

- Added composite constraint

PATTERNS

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

MATH

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

TEST SUITE

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

MISCELLANEA

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

- Upgraded to use Doxygen 1.3.4

Release 0.3.3 - September 3rd, 2003

MONTE CARLO FRAMEWORK

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

FINITE DIFFERENCE FRAMEWORK

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

LATTICE FRAMEWORK

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

PRICING ENGINES FRAMEWORK

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

YIELD TERM STRUCTURE

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

FIXED INCOME

- Up-front and in-arrear indexed coupon.
- Specific implementation of compound forward rate from zero yield.

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

OPTIMIZATION FRAMEWORK

- Solvers now take any function. ObjectiveFunction disappeared.

PATTERNS

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

DATE AND CALENDARS

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

MATH

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

- Added DiscrepancyStatistic that inherits from SequenceStatistic and extends it with the calculation of L2-discrepancy.
- HStatistics.
- Added first and second derivative of cubic splines.

RISK MEASURES

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

MISCELLANEA

- QuEP 9 "generic disposable objects" implemented.
- Added test suite.
- Dataformatters extended to format long integers, Ordinal numerals, power of two formatting.
- Exercise class adopted.
- Added user configuration section.
- Inhibited automatic conversion of Handle<T> to RelinkableHandle<T>.
- Diffusion process extended.
- Added strikeSensitivity to the Greeks.
- BS does handle $t=0.0$ and $\sigma=0.0$.
- TimeGrid has been reworked.
- Added payoff file for Payoff classes. Added Cash-Or-Nothing and Asset-Or-Nothing payoff classes.
- Upgraded to use Doxygen 1.3.

Release 0.3.1 - February 4th, 2003

FINITE DIFFERENCE FRAMEWORK

- partially implemented QuEP 2 (<http://quantlib.org/quep.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.

1.9 Additional resources

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

Additional resources available from the above site include:

- current news (http://sourceforge.net/news/?group_id=12740);
- the QuantLib mailing lists and forums (<http://quantlib.org/maillinglists.shtml>);
- the QuantLib programming style guidelines (<http://quantlib.org/style.shtml>);
- a link to the QuantLib project page on SourceForge.net (<http://sourceforge.net/projects/quantlib>);
- links to pages for bug reports (http://sourceforge.net/tracker/?group_id=12740&atid=112740), patch submissions (http://sourceforge.net/tracker/?group_id=12740&atid=312740), and feature requests (http://sourceforge.net/tracker/?group_id=12740&atid=362740);
- a page (<http://quantlib.org/extensions.shtml>) about how to use QuantLib in other languages/platforms;
- QuantLib web-site statistics (http://sourceforge.net/project/stats/?group_id=12740);
- as well as links to additional quantitative finance resources.

1.10 The QuantLib Group

1.10.1 Authors

The QuantLib Group members are:

- Ferdinando Ametrano, Monte Paschi Asset Management sgr, administrator
- Luigi Ballabio, StatPro Italia srl, administrator
- Mario Aleppo, StatPro Italia srl
- Nicolas Di Césaré
- Dirk Eddelbuettel
- Neil Firth, Mathematical Institute, University of Oxford
- André Louw, Decillion Pty
- Marco Marchioro, StatPro Italia srl
- Sadruddin Rejeb
- Niels Elken Sønderby
- Enrico Sirola, StatPro Italia srl
- Liguó Song

1.10.2 Contributors

We gratefully acknowledge contributions from Xavier Abulker, Toyin Akin, James Battle, Christopher Baus, Thomas Becker, Adolfo Benin, David Binderman, Jon Davidson, Daniele De Francesco, Matteo Gallivanoni, Roman Gitlin, Tomoya Kawanishi, Gilbert Pepper, Walter Penschke, Gianni Piolanti, Peter Schmitteckert, Maxim Sokolov, David Schwartz, Bernd Johannes Wuebben, and Jeff Yu.

1.11 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.

1.11.1 Comments on Copyright and License

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

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

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

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

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

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

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

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

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

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

Chapter 2

QuantLib Module Index

2.1 QuantLib Modules

Here is a list of all modules:

Numeric types	75
Currencies and FX rates	77
Date and time calculations	80
Calendars	83
Day counters	85
Pricing engines	86
Asian option engines	87
Barrier option engines	88
Basket option engines	89
Cap/floor engines	90
Cliquet option engines	91
Forward option engines	92
Quanto option engines	93
Swaption engines	94
Vanilla option engines	95
Finite-differences framework	96
Short-rate modelling framework	102
Financial instruments	105
Lattice methods	108
Math tools	111
Monte Carlo framework	113
Design patterns	119
Term structures	120
Utilities	121
QuantLib macros	123
Generic macros	124
Math functions	125
Numeric limits	126
Time functions	127
String functions	128
Character functions	129
Min and max functions	130

Template capabilities	131
Iterator support	132

Chapter 3

QuantLib Hierarchical Index

3.1 QuantLib Class Hierarchy

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

AcyclicVisitor	136
BPSBasketCalculator	221
BPSCalculator	222
AmericanPayoffAtExpiry	143
AmericanPayoffAtHit	144
Arguments	154
CapFloor::arguments	243
Option::arguments	635
MultiAssetOption::arguments	593
BasketOption::arguments	176
OneAssetOption::arguments	622
Swaption::arguments	750
SimpleSwap::arguments	713
Swaption::arguments	750
Array	156
ArrayFormatter	159
Average	165
Barrier	167
BarrierOption::arguments	170
BinomialDistribution	185
BivariateCumulativeNormalDistribution	189
BlackFormula	194
BlackKarasinski::Dynamics	196
BoundaryCondition	217
BoundaryCondition< TridiagonalOperator >	217
DirichletBC	324
NeumannBC	601
BoxMullerGaussianRng	220
Bridge	224
Bridge< Calendar, CalendarImpl >	224
Calendar	232
Beijing	180

Budapest	228
Copenhagen	284
Germany	439
Helsinki	447
HongKong	454
Italy	500
Johannesburg	505
JointCalendar	506
NullCalendar	611
Oslo	639
Riyadh	684
Seoul	696
Singapore	717
Stockholm	739
Sydney	755
Taiwan	757
TARGET	758
Tokyo	766
Toronto	768
UnitedKingdom	787
UnitedStates	789
Warsaw	804
Wellington	806
Zurich	825
Bridge< Constraint, ConstraintImpl >	224
Constraint	277
BoundaryConstraint	219
CompositeConstraint	269
NoConstraint	605
PositiveConstraint	658
Bridge< DayCounter, DayCounterImpl >	224
DayCounter	316
Actual360	133
Actual365Fixed	134
ActualActual	135
OneDayCounter	626
SimpleDayCounter	709
Thirty360	762
Bridge< Interpolation, InterpolationImpl >	224
Interpolation	487
CubicSpline	297
MonotonicCubicSpline	586
NaturalCubicSpline	599
NaturalMonotonicCubicSpline	600
LinearInterpolation	528
LogLinearInterpolation	541
Bridge< Interpolation2D, Interpolation2DImpl >	224
Interpolation2D	488
BicubicSpline	183
BilinearInterpolation	184
Bridge< Parameter, ParameterImpl >	224
Parameter	640

ConstantParameter	276
NullParameter	612
PiecewiseConstantParameter	650
TermStructureFittingParameter	760
ExtendedCoxIngersollRoss::FittingParameter	378
G2::FittingParameter	421
HullWhite::FittingParameter	458
BrownianBridge	226
CalendarImpl	237
Calendar::WesternImpl	236
CLGaussianRng	257
CliquetOption::arguments	260
combining_iterator	266
Composite	268
CompoundingRuleFormatter	274
ConstraintImpl	278
ContinuousAveragingAsianOption::arguments	281
CostFunction	285
LeastSquareFunction	521
coupling_iterator	286
CovarianceDecomposition	290
CoxIngersollRoss::Dynamics	293
ExtendedCoxIngersollRoss::Dynamics	377
Cubic	296
CumulativeBinomialDistribution	299
CumulativeNormalDistribution	300
CumulativePoissonDistribution	301
CuriouslyRecurringTemplate	302
Solver1D	726
CuriouslyRecurringTemplate< Bisection >	302
Solver1D< Bisection >	726
Bisection	188
CuriouslyRecurringTemplate< Brent >	302
Solver1D< Brent >	726
Brent	223
CuriouslyRecurringTemplate< FalsePosition >	302
Solver1D< FalsePosition >	726
FalsePosition	383
CuriouslyRecurringTemplate< Newton >	302
Solver1D< Newton >	726
Newton	602
CuriouslyRecurringTemplate< NewtonSafe >	302
Solver1D< NewtonSafe >	726
NewtonSafe	603
CuriouslyRecurringTemplate< Ridder >	302
Solver1D< Ridder >	726
Ridder	683
CuriouslyRecurringTemplate< Secant >	302
Solver1D< Secant >	726
Secant	692

Currency	303
ARSCurrency	160
ATSCurrency	162
AUDCurrency	163
BDTCurrency	178
BEFCurrency	179
BGLCurrency	182
BRLCurrency	225
BYRCurrency	229
CADCurrency	230
CHFCurrency	255
CLPCurrency	263
CNYCurrency	264
COPCurrency	283
CYPCurrency	308
CZKCurrency	309
DEMCurrency	320
DKKCurrency	347
EEKCurrency	356
ESPCurrency	363
EURCurrency	365
FIMCurrency	394
FRFCurrency	417
GBPCurrency	428
GRDCurrency	442
HKDCurrency	453
HUFCurrency	455
IEPCurrency	461
ILSCurrency	462
INRCurrency	477
IQDCurrency	497
IRRCurrency	498
ISKCurrency	499
ITLCurrency	502
JPYCurrency	507
KRWCurrency	513
KWDCurrency	514
LTLCurrency	543
LUFCurrency	544
LVLCurrency	545
MTLCurrency	590
MXNCurrency	598
NLGCurrency	604
NOKCurrency	606
NPRCurrency	609
NZDCurrency	615
PKRCurrency	654
PLNCurrency	656
PTECurrency	664
ROLCurrency	685
SARCurrency	690
SEKCurrency	695
SGDCurrency	702

SITCurrency	721
SKKCurrency	723
THBCurrency	761
TRLCurrency	780
TTDCurrency	781
TWDCurrency	782
USDCurrency	794
VEBCurrency	801
ZARCurrency	817
CurrencyFormatter	307
Date	310
DateFormatter	315
DayCounterImpl	318
DecimalFormatter	319
DiscreteAveragingAsianOption::arguments	332
DiscretizedAsset	336
DiscretizedDiscountBond	339
DiscretizedOption	340
Disposable	342
DividendVanillaOption::arguments	345
EndCriteria	357
Error	361
ErrorFunction	362
EuroFormatter	367
ExchangeRate	370
Exercise	374
EarlyExercise	355
AmericanExercise	142
BermudanExercise	181
EuropeanExercise	368
Extrapolator	381
BlackVolTermStructure	211
BlackVarianceTermStructure	207
BlackVarianceCurve	203
BlackVarianceSurface	205
ImpliedVolTermStructure	466
BlackVolatilityTermStructure	209
BlackConstantVol	192
LocalVolTermStructure	538
LocalConstantVol	533
LocalVolCurve	534
LocalVolSurface	536
YieldTermStructure	810
AffineTermStructure	139
DiscountStructure	327
DiscountCurve	325
ExtendedDiscountCurve	379
ImpliedTermStructure	464
FlatForward	399
ForwardRateStructure	408
CompoundForward	271
ForwardSpreadedTermStructure	410

PiecewiseFlatForward	651
ZeroYieldStructure	823
DriftTermStructure	352
QuantoTermStructure	671
ZeroCurve	819
ZeroSpreadedTermStructure	821
Factorial	382
FaureRsg	384
FdBermudanOption	386
FdDividendAmericanOption	389
FdDividendShoutOption	390
filtering_iterator	393
FiniteDifferenceModel	395
ForwardOptionArguments	406
FrequencyFormatter	416
GammaFunction	423
GaussianStatistics	425
GeneralStatistics	430
GenericRiskStatistics	435
HaltonRsg	444
Handle	445
History	448
History::const_iterator	451
History::Entry	452
HullWhite::Dynamics	457
ICGaussianRng	459
ICGaussianRsg	460
IncrementalStatistics	470
IntegerFormatter	481
InterestRate	483
InterestRateFormatter	486
Interpolation2DImpl	490
Interpolation2D::templateImpl	489
InterpolationImpl	492
Interpolation::templateImpl	491
InverseCumulativeNormal	493
InverseCumulativePoisson	494
InverseCumulativeRng	495
InverseCumulativeRsg	496
KnuthUniformRng	511
KronrodIntegral	512
LeastSquareProblem	522
LecuyerUniformRng	523
LexicographicalView	525
Linear	527
LineSearch	529
ArmijoLineSearch	155
LogLinear	540
lowest_category_iterator	542
MakeSchedule	546
Matrix	547
MatrixFormatter	551
McPricer	573

McPricer< MultiAsset< PseudoRandom > >	573
McEverest	568
McHimalaya	569
McMaxBasket	570
McPagoda	571
McPricer< SingleAsset< PseudoRandom > >	573
McCliquetOption	557
McDiscreteArithmeticAPO	562
McDiscreteArithmeticASO	563
McPerformanceOption	572
McSimulation	574
McSimulation< MultiAsset< RNG >, S >	574
MCBasketEngine	555
McSimulation< SingleAsset< RNG >, S >	574
MCBarrierEngine	553
MCDiscreteAveragingAsianEngine	564
MCDiscreteArithmeticAPEngine	560
MCDiscreteGeometricAPEngine	566
MCVanillaEngine	576
MCDigitalEngine	558
MCEuropeanEngine	567
MersenneTwisterUniformRng	578
MixedScheme	581
CrankNicolson	295
ExplicitEuler	375
ImplicitEuler	463
Money	583
MoneyFormatter	585
MonteCarloModel	587
MoroInverseCumulativeNormal	589
MultiCubicSpline	595
MultiPath	596
MultiPathGenerator	597
NonLinearLeastSquare	607
NormalDistribution	608
Null	610
NumericalMethod	613
Lattice	515
BlackScholesLattice	199
Lattice2D	517
TwoFactorModel::ShortRateTree	785
OneFactorModel::ShortRateTree	630
Observable	616
AffineModel	138
G2	419
OneFactorAffineModel	627
CoxIngersollRoss	291
ExtendedCoxIngersollRoss	376
Vasicek	799
HullWhite	456
BaseTermStructure	172

BlackVolTermStructure	211
CapletVolatilityStructure	246
CapletConstantVolatility	245
CapVolatilityStructure	248
CapVolatilityVector	250
LocalVolTermStructure	538
SwaptionVolatilityStructure	753
SwaptionVolatilityMatrix	752
YieldTermStructure	810
BlackModel	197
CalibrationHelper	238
CashFlow	252
Coupon	288
FixedRateCoupon	397
FloatingRateCoupon	401
IndexedCoupon	474
InArrearIndexedCoupon	468
UpFrontIndexedCoupon	792
ParCoupon	642
Short< ParCoupon >	704
SimpleCashFlow	708
Index	473
Xibor	807
AUDLibor	164
CADLibor	231
CHFLibor	256
Euribor	366
GBPLibor	429
JPYLibor	508
USDLibor	795
ZARLibor	818
LazyObject	519
AffineTermStructure	139
Instrument	478
Bond	214
FixedCouponBond	396
CapFloor	241
Cap	240
Collar	265
Floor	403
Option	634
MultiAssetOption	591
BasketOption	175
OneAssetOption	619
OneAssetStrikedOption	624
BarrierOption	168
CliquetOption	258
ContinuousAveragingAsianOption	279
DiscreteAveragingAsianOption	330
VanillaOption	797
DividendVanillaOption	343
EuropeanOption	369

ForwardVanillaOption	412
QuantoVanillaOption	673
QuantoForwardVanillaOption	667
Swaption	749
Stock	738
Swap	745
SimpleSwap	711
PiecewiseFlatForward	651
Link	531
PricingEngine	659
GenericEngine	433
GenericModelEngine	434
GenericEngine< Arguments, Results >	433
GenericModelEngine< ShortRateModel, Arguments, Results >	434
LatticeShortRateModelEngine	518
GenericEngine< BarrierOption::arguments, BarrierOption::results >	433
BarrierOption::engine	171
AnalyticBarrierEngine	145
MCBarrierEngine	553
GenericEngine< BasketOption::arguments, BasketOption::results >	433
BasketOption::engine	177
MCAmericanBasketEngine	552
MCBasketEngine	555
StulzEngine	742
GenericEngine< CapFloor::arguments, CapFloor::results >	433
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >	434
AnalyticCapFloorEngine	146
GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results >	434
BlackCapFloorEngine	191
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >	434
LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >	518
TreeCapFloorEngine	772
GenericEngine< CliquetOption::arguments, CliquetOption::results >	433
CliquetOption::engine	261
AnalyticCliquetEngine	147
AnalyticPerformanceEngine	153
GenericEngine< ContinuousAveragingAsianOption::arguments, ContinuousAveragingAsianOption::results >	433
ContinuousAveragingAsianOption::engine	282
AnalyticContinuousGeometricAveragePriceAsianEngine	148
GenericEngine< DiscreteAveragingAsianOption::arguments, DiscreteAveragingAsianOption::results >	433
DiscreteAveragingAsianOption::engine	333
AnalyticDiscreteGeometricAveragePriceAsianEngine	150
MCDiscreteAveragingAsianEngine	564
GenericEngine< DividendVanillaOption::arguments, DividendVanillaOption::results >	433
DividendVanillaOption::engine	346
AnalyticDividendEuropeanEngine	151
GenericEngine< ForwardOptionArguments< ArgumentsType >, ResultsType >	433

ForwardEngine	405
ForwardPerformanceEngine	407
GenericEngine< QuantoOptionArguments< ArgumentsType >, QuantoOption- Results< ResultsType > >	433
QuantoEngine	665
GenericEngine< Swaption::arguments, Swaption::results >	433
GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results >	434
BlackSwaptionEngine	202
GenericModelEngine< G2, Swaption::arguments, Swaption::results >	434
G2SwaptionEngine	422
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swap- tion::results >	434
JamshidianSwaptionEngine	503
GenericModelEngine< ShortRateModel, Swaption::arguments, Swap- tion::results >	434
LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >	518
TreeSwaptionEngine	773
GenericEngine< VanillaOption::arguments, VanillaOption::results >	433
VanillaOption::engine	798
AnalyticDigitalAmericanEngine	149
AnalyticEuropeanEngine	152
BaroneAdesiWhaleyApproximationEngine	166
BinomialVanillaEngine	187
Bjerk Sund Stensland ApproximationEngine	190
IntegralEngine	482
JumpDiffusionEngine	509
JuQuadraticApproximationEngine	510
MCVanillaEngine	576
Quote	675
CompositeQuote	270
DerivedQuote	323
SimpleQuote	710
RateHelper	680
DepositRateHelper	321
FraRateHelper	414
FuturesRateHelper	418
SwapRateHelper	747
ShortRateModel	705
OneFactorModel	628
BlackKarasinski	195
OneFactorAffineModel	627
TwoFactorModel	783
G2	419
StochasticProcess	735
BlackScholesProcess	200
Merton76Process	579
GeometricBrownianMotionProcess	438
OrnsteinUhlenbeckProcess	637
SquareRootProcess	728
TermStructureConsistentModel	759
BlackKarasinski	195
ExtendedCoxIngersollRoss	376

G2	419
HullWhite	456
Observer	617
BaseTermStructure	172
BlackModel	197
CalibrationHelper	238
CompositeQuote	270
DerivedQuote	323
GenericModelEngine	434
GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results > . . .	434
GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results > . . .	434
GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results > . . .	434
GenericModelEngine< G2, Swaption::arguments, Swaption::results >	434
GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swap- tion::results >	434
GenericModelEngine< ShortRateModel, Arguments, Results >	434
GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results > .	434
GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results > .	434
IndexedCoupon	474
LazyObject	519
Link	531
ParCoupon	642
RateHelper	680
ShortRateModel	705
StochasticProcess	735
Xibor	807
OneFactorModel::ShortRateDynamics	629
OptimizationMethod	632
ConjugateGradient	275
Simplex	715
SteepestDescent	730
OptionTypeFormatter	636
ParameterImpl	641
Path	644
PathGenerator	645
PathPricer	646
PathPricer< MultiPath >	646
PathPricer< Path >	646
Payoff	647
TypePayoff	786
StrikedTypePayoff	740
AssetOrNothingPayoff	161
CashOrNothingPayoff	253
GapPayoff	424
PercentageStrikePayoff	648
PlainVanillaPayoff	655
SuperSharePayoff	743
Period	649
PoissonDistribution	657
PrimeNumbers	660
Problem	661
processing_iterator	662
QuantoOptionArguments	669

QuantoOptionResults	670
RamdomizedLDS	676
RandomSequenceGenerator	678
RateFormatter	679
Results	682
Greeks	443
MultiAssetOption::results	594
OneAssetOption::results	623
MoreGreeks	588
OneAssetOption::results	623
Value	796
CapFloor::results	244
MultiAssetOption::results	594
OneAssetOption::results	623
SimpleSwap::results	714
Swaption::results	751
Rounding	686
CeilingTruncation	254
ClosestRounding	262
DownRounding	349
FloorTruncation	404
UpRounding	793
SalvagingAlgorithm	688
Sample	689
Schedule	691
SegmentIntegral	694
SequenceFormatter	697
SequenceStatistics	698
SequenceStatistics< Statistics >	698
DiscrepancyStatistics	329
Short	703
SingleAssetOption	718
DiscreteGeometricAPO	334
DiscreteGeometricASO	335
FdBsmOption	387
FdEuropean	391
FdStepConditionOption	392
FdAmericanOption	385
Singleton	720
Singleton< ExchangeRateManager >	720
ExchangeRateManager	372
Singleton< IndexManager >	720
IndexManager	476
Singleton< SeedGenerator >	720
SeedGenerator	693
Singleton< Settings >	720
Settings	700
SizeFormatter	722
SobolRsg	724
StatsHolder	729
step_iterator	731

StepCondition	732
AmericanCondition	141
ShoutCondition	707
StepCondition< Array >	732
stepping_iterator	733
StochasticProcess::discretization	737
EulerDiscretization	364
StringFormatter	741
SVD	744
SymmetricSchurDecomposition	756
TimeBasket	764
TimeGrid	765
TrapezoidIntegral	769
SimpsonIntegral	716
Tree	771
BinomialTree	186
EqualJumpsBinomialTree	359
CoxRossRubinstein	294
Trigeorgis	777
EqualProbabilitiesBinomialTree	360
AdditiveEQPBinomialTree	137
JarrowRudd	504
LeisenReimer	524
Tian	763
TrinomialTree	779
TridiagonalOperator	774
BSMOperator	227
DMinus	348
DPlus	350
DPlusDMinus	351
DZero	354
OneFactorOperator	631
TridiagonalOperator::TimeSetter	776
TrinomialBranching	778
TwoFactorModel::ShortRateDynamics	784
Vasicek::Dynamics	800
Visitor	802
Visitor< BlackConstantVol >	802
Visitor< BlackVarianceCurve >	802
Visitor< BlackVolTermStructure >	802
Visitor< CashFlow >	802
BPSBasketCalculator	221
BPSCalculator	222
Visitor< Coupon >	802
BPSBasketCalculator	221
BPSCalculator	222
Visitor< FixedRateCoupon >	802
BPSBasketCalculator	221
VolatilityFormatter	803
WeekdayFormatter	805
XiborManager	809

Chapter 4

QuantLib Class Index

4.1 QuantLib Class List

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

Actual360 (Actual/360 day count convention)	133
Actual365Fixed (Actual/365 (Fixed) day count convention)	134
ActualActual (Actual/Actual day count)	135
AcyclicVisitor (Degenerate base class for the Acyclic Visitor pattern)	136
AdditiveEQPBinomialTree (Additive equal probabilities binomial tree)	137
AffineModel (Affine model class)	138
AffineTermStructure (Term-structure implied by an affine model)	139
AmericanCondition (American exercise condition)	141
AmericanExercise (American exercise)	142
AmericanPayoffAtExpiry	143
AmericanPayoffAtHit	144
AnalyticBarrierEngine (Pricing engine for barrier options using analytical formulae) . .	145
AnalyticCapFloorEngine (Analytic engine for cap/floor)	146
AnalyticCliquetEngine (Pricing engine for Cliquet options using analytical formulae) .	147
AnalyticContinuousGeometricAveragePriceAsianEngine (Pricing engine for European continuous geometric average price Asian)	148
AnalyticDigitalAmericanEngine	149
AnalyticDiscreteGeometricAveragePriceAsianEngine (Pricing engine for European dis- crete geometric average price Asian)	150
AnalyticDividendEuropeanEngine (Analytic pricing engine for European options with discrete dividends)	151
AnalyticEuropeanEngine (Pricing engine for European vanilla options using analytical formulae)	152
AnalyticPerformanceEngine (Pricing engine for performance options using analytical formulae)	153
Arguments (Base class for generic argument groups)	154
ArmijoLineSearch (Armijo line search)	155
Array (1-D array used in linear algebra)	156
ArrayFormatter (Format arrays for output)	159
ARSCurrency (Argentinian peso)	160
AssetOrNothingPayoff (Binary asset-or-nothing payoff)	161
ATSCurrency (Austrian shilling)	162
AUDCurrency (Australian dollar)	163

AUDLibor (AUD Libor index, also known as SIBOR)	164
Average (Placeholder for enumerated averaging types)	165
BaroneAdesiWhaleyApproximationEngine	166
Barrier (Placeholder for enumerated barrier types)	167
BarrierOption (Barrier option on a single asset)	168
BarrierOption::arguments (Arguments for barrier option calculation)	170
BarrierOption::engine (Barrier engine base class)	171
BaseTermStructure (Basic term-structure functionality)	172
BasketOption (Basket option on a number of assets)	175
BasketOption::arguments (Arguments for basket option calculation)	176
BasketOption::engine (Basket option engine base class)	177
BDTCurrency (Bangladesh taka)	178
BEFCurrency (Belgian franc)	179
Beijing (Beijing calendar)	180
BermudanExercise (Bermudan exercise)	181
BGLCurrency (Bulgarian lev)	182
BicubicSpline	183
BilinearInterpolation (Bilinear interpolation between discrete points)	184
BinomialDistribution (Binomial probability distribution function)	185
BinomialTree (Binomial tree base class)	186
BinomialVanillaEngine (Pricing engine for vanilla options using binomial trees)	187
Bisection (Bisection 1-D solver)	188
BivariateCumulativeNormalDistribution (Cumulative bivariate normal distribution function)	189
BjerkstrandStenslandApproximationEngine	190
BlackCapFloorEngine (Black-formula cap/floor engine)	191
BlackConstantVol (Constant Black volatility, no time-strike dependence)	192
BlackFormula (Black-formula calculator)	194
BlackKarasinski (Standard Black-Karasinski model class)	195
BlackKarasinski::Dynamics (Short-rate dynamics in the Black-Karasinski model)	196
BlackModel (Black-model for vanilla interest-rate derivatives)	197
BlackScholesLattice (Simple binomial lattice approximating the Black-Scholes model)	199
BlackScholesProcess (Black-Scholes stochastic process)	200
BlackSwaptionEngine (Black-formula swaption engine)	202
BlackVarianceCurve (Black volatility curve modelled as variance curve)	203
BlackVarianceSurface (Black volatility surface modelled as variance surface)	205
BlackVarianceTermStructure (Black variance term structure)	207
BlackVolatilityTermStructure (Black-volatility term structure)	209
BlackVolTermStructure (Black-volatility term structure)	211
Bond (Base bond class)	214
BoundaryCondition (Abstract boundary condition class for finite difference problems)	217
BoundaryConstraint (Constraint imposing all arguments to be in [low,high])	219
BoxMullerGaussianRng (Gaussian random number generator)	220
BPSBasketCalculator	221
BPSCalculator (Basis point sensitivity (BPS) calculator)	222
Brent (Brent 1-D solver)	223
Bridge (The Bridge pattern made explicit)	224
BRLCurrency (Brazilian real)	225
BrownianBridge (Builds Wiener process paths using Gaussian variates)	226
BSMOperator (Black-Scholes-Merton differential operator)	227
Budapest (Budapest calendar)	228
BYRCurrency (Belarussian ruble)	229
CADCurrency (Canadian dollar)	230
CADLibor (CAD Libor index, also known as CDOR)	231

Calendar (calendar class)	232
Calendar::WesternImpl (Partial calendar implementation)	236
CalendarImpl (Abstract base class for calendar implementations)	237
CalibrationHelper (Liquid market instrument used during calibration)	238
Cap (Concrete cap class)	240
CapFloor (Base class for cap-like instruments)	241
CapFloor::arguments (Arguments for cap/floor calculation)	243
CapFloor::results (Results from cap/floor calculation)	244
CapletConstantVolatility (Constant caplet volatility, no time-strike dependence)	245
CapletVolatilityStructure (Caplet/floorlet forward-volatility structure)	246
CapVolatilityStructure (Cap/floor term-volatility structure)	248
CapVolatilityVector (Cap/floor at-the-money term-volatility vector)	250
CashFlow (Base class for cash flows)	252
CashOrNothingPayoff (Binary cash-or-nothing payoff)	253
CeilingTruncation (Ceiling truncation)	254
CHFCurrency (Swiss franc)	255
CHFLibor (CHF Libor index, also known as ZIBOR)	256
CLGaussianRng (Gaussian random number generator)	257
CliquetOption (Cliquet (Ratchet) option)	258
CliquetOption::arguments (Arguments for cliquet option calculation)	260
CliquetOption::engine (Cliquet engine base class)	261
ClosestRounding (Closest rounding)	262
CLPCurrency (Chilean peso)	263
CNYCurrency (Chinese yuan)	264
Collar (Concrete collar class)	265
combining_iterator (Iterator mapping a function to a set of underlying sequences)	266
Composite (Composite pattern)	268
CompositeConstraint (Constraint enforcing both given sub-constraints)	269
CompositeQuote (Market element whose value depends on two other market element)	270
CompoundForward (Compound-forward structure)	271
CompoundingRuleFormatter (Formats compounding rule for output)	274
ConjugateGradient (Multi-dimensional Conjugate Gradient class)	275
ConstantParameter (Standard constant parameter $a(t) = a$)	276
Constraint (Base constraint class)	277
ConstraintImpl (Base class for constraint implementations)	278
ContinuousAveragingAsianOption (Continuous-averaging Asian option)	279
ContinuousAveragingAsianOption::arguments (Extra arguments for single-asset continuous-average Asian option)	281
ContinuousAveragingAsianOption::engine (Continuous-averaging Asian engine base class)	282
COPCurrency (Colombian peso)	283
Copenhagen (Copenhagen calendar)	284
CostFunction (Cost function abstract class for optimization problem)	285
coupling_iterator (Iterator mapping a function to a pair of underlying sequences)	286
Coupon (coupon accruing over a fixed period)	288
CovarianceDecomposition	290
CoxIngersollRoss (Cox-Ingersoll-Ross model class)	291
CoxIngersollRoss::Dynamics (Dynamics of the short-rate under the Cox-Ingersoll-Ross model)	293
CoxRossRubinstein (Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree)	294
CrankNicolson (Crank-Nicolson scheme for finite difference methods)	295
Cubic (Cubic-spline interpolation traits)	296
CubicSpline (Cubic spline interpolation between discrete points)	297
CumulativeBinomialDistribution (Cumulative binomial distribution function)	299

CumulativeNormalDistribution (Cumulative normal distribution function)	300
CumulativePoissonDistribution (Cumulative Poisson distribution function)	301
CuriouslyRecurringTemplate (Support for the curiously recurring template pattern) . .	302
Currency (Currency specification)	303
CurrencyFormatter (Format currencies for output)	307
CYPCurrency (Cyprus pound)	308
CZKCurrency (Czech koruna)	309
Date (Concrete date class)	310
DateFormatter (Formats dates for output)	315
DayCounter (Day counter class)	316
DayCounterImpl (Abstract base class for day counter implementations)	318
DecimalFormatter (Formats real numbers for output)	319
DEMCurrency (Deutsche mark)	320
DepositRateHelper (Deposit rate)	321
DerivedQuote (Market element whose value depends on another market element) . . .	323
DirichletBC (Neumann boundary condition (i.e., constant value))	324
DiscountCurve (Term structure based on loglinear interpolation of discount factors) . .	325
DiscountStructure (Discount factor term structure)	327
DiscrepancyStatistics (Statistic tool for sequences with discrepancy calculation)	329
DiscreteAveragingAsianOption (Discrete-averaging Asian option)	330
DiscreteAveragingAsianOption::arguments (Extra arguments for single-asset discrete- average Asian option)	332
DiscreteAveragingAsianOption::engine (Discrete-averaging Asian engine base class) . .	333
DiscreteGeometricAPO (Discrete geometric average-price Asian option (European style))	334
DiscreteGeometricASO (Discrete geometric average-strike Asian option (European style))	335
DiscretizedAsset (Discretized asset class used by numerical methods)	336
DiscretizedDiscountBond (Useful discretized discount bond asset)	339
DiscretizedOption (Discretized option on a given asset)	340
Disposable (Generic disposable object with move semantics)	342
DividendVanillaOption (Single-asset vanilla option (no barriers) with discrete dividends)	343
DividendVanillaOption::arguments (Arguments for dividend vanilla option calculation)	345
DividendVanillaOption::engine (Dividend vanilla option engine base class)	346
DKKCurrency (Danish krone)	347
DMinus (D_- matricial representation)	348
DownRounding (Down-rounding)	349
DPlus (D_+ matricial representation)	350
DPlusDMinus (D_+D_- matricial representation)	351
DriftTermStructure (Drift term structure)	352
DZero (D_0 matricial representation)	354
EarlyExercise (Early-exercise base class)	355
EEKCurrency (Estonian kroon)	356
EndCriteria (Criteria to end optimization process)	357
EqualJumpsBinomialTree (Base class for equal jumps binomial tree)	359
EqualProbabilitiesBinomialTree (Base class for equal probabilities binomial tree)	360
Error (Base error class)	361
ErrorFunction (Error function)	362
ESPCurrency (Spanish peseta)	363
EulerDiscretization (Euler discretization for stochastic processes)	364
EURCurrency (European Euro)	365
Euribor (Euribor index)	366
EuroFormatter (Formats amounts in Euro for output)	367
EuropeanExercise (European exercise)	368
EuropeanOption (European option on a single asset)	369
ExchangeRate (Exchange rate between two currencies)	370

ExchangeRateManager (Exchange-rate repository)	372
Exercise (Base exercise class)	374
ExplicitEuler (Forward Euler scheme for finite difference methods)	375
ExtendedCoxIngersollRoss (Extended Cox-Ingersoll-Ross model class)	376
ExtendedCoxIngersollRoss::Dynamics (Short-rate dynamics in the extended Cox-Ingersoll-Ross model)	377
ExtendedCoxIngersollRoss::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	378
ExtendedDiscountCurve (Term structure based on loglinear interpolation of discount factors)	379
Extrapolator (Base class for classes possibly allowing extrapolation)	381
Factorial (Factorial numbers calculator)	382
FalsePosition (False position 1-D solver)	383
FaureRsg (Faure low-discrepancy sequence generator)	384
FdAmericanOption (American option)	385
FdBermudanOption (Bermudan option)	386
FdBsmOption (Black-Scholes-Merton option priced numerically)	387
FdDividendAmericanOption (American option with discrete dividends)	389
FdDividendShoutOption (Shout option with dividends)	390
FdEuropean (Example of European option calculated using finite differences)	391
FdStepConditionOption (option executing additional code at each time step)	392
filtering_iterator (Iterator filtering undesired data)	393
FIMCurrency (Finnish markka)	394
FiniteDifferenceModel (Generic finite difference model)	395
FixedCouponBond (Fixed-coupon bond)	396
FixedRateCoupon (Coupon paying a fixed interest rate)	397
FlatForward (Flat interest-rate curve)	399
FloatingRateCoupon (Coupon paying a variable rate)	401
Floor (Concrete floor class)	403
FloorTruncation (Floor truncation)	404
ForwardEngine (Forward engine base class)	405
ForwardOptionArguments (Arguments for forward (strike-resetting) option calculation)	406
ForwardPerformanceEngine (Forward performance engine)	407
ForwardRateStructure (Forward rate term structure)	408
ForwardSpreadedTermStructure (Term structure with added spread on the instantaneous forward rate)	410
ForwardVanillaOption (Forward version of a vanilla option)	412
FraRateHelper (Forward rate agreement)	414
FrequencyFormater (Formats frequency for output)	416
FRFCurrency (French franc)	417
FuturesRateHelper (Interest-rate futures)	418
G2 (Two-additive-factor gaussian model class)	419
G2::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	421
G2SwaptionEngine (Swaption priced by means of the Black formula)	422
GammaFunction (Gamma function class)	423
GapPayoff (Binary gap payoff)	424
GaussianStatistics (Statistics tool for gaussian-assumption risk measures)	425
GBPCurrency (British pound sterling)	428
GBPLibor (GBP Libor index)	429
GeneralStatistics (Statistics tool)	430
GenericEngine (Template base class for option pricing engines)	433
GenericModelEngine (Base class for some pricing engine on a particular model)	434
GenericRiskStatistics (Empirical-distribution risk measures)	435
GeometricBrownianMotionProcess (Geometric brownian motion process)	438

Germany (German calendars)	439
GRDCurrency (Greek drachma)	442
Greeks (Additional option results)	443
HaltonRsg (Halton low-discrepancy sequence generator)	444
Handle (Globally accessible relinkable pointer)	445
Helsinki (Helsinki calendar)	447
History (Container for historical data)	448
History::const_iterator (Random access iterator on history entries)	451
History::Entry (Single datum in history)	452
HKDCurrency (Honk Kong dollar)	453
HongKong (Hong Kong calendar)	454
HUFCurrency (Hungarian forint)	455
HullWhite (Single-factor Hull-White (extended Vasicek) model class)	456
HullWhite::Dynamics (Short-rate dynamics in the Hull-White model)	457
HullWhite::FittingParameter (Analytical term-structure fitting parameter $\varphi(t)$)	458
ICGaussianRng (Inverse cumulative Gaussian random number generator)	459
ICGaussianRsg (Inverse cumulative Gaussian random sequence generator)	460
IEPCurrency (Irish punt)	461
ILSCurrency (Israeli shekel)	462
ImplicitEuler (Backward Euler scheme for finite difference methods)	463
ImpliedTermStructure (Implied term structure at a given date in the future)	464
ImpliedVolTermStructure (Implied vol term structure at a given date in the future)	466
InArrearIndexedCoupon (In-arrear floating-rate coupon)	468
IncrementalStatistics (Statistics tool based on incremental accumulation)	470
Index (Purely virtual base class for indexes)	473
IndexedCoupon (Base indexed coupon class)	474
IndexManager (Global repository for past index fixings)	476
INRCurrency (Indian rupee)	477
Instrument (Abstract instrument class)	478
IntegerFormatter (Formats integers for output)	481
IntegralEngine	482
InterestRate (Concrete interest rate class)	483
InterestRateFormatter (Formats interest rates for output)	486
Interpolation (Base class for 1-D interpolations)	487
Interpolation2D (Base class for 2-D interpolations)	488
Interpolation2D::templateImpl (Basic template implementation)	489
Interpolation2DImpl (Abstract base class for 2-D interpolation implementations)	490
Interpolation::templateImpl (Basic template implementation)	491
InterpolationImpl (Abstract base class for interpolation implementations)	492
InverseCumulativeNormal (Inverse cumulative normal distribution function)	493
InverseCumulativePoisson (Inverse cumulative Poisson distribution function)	494
InverseCumulativeRng (Inverse cumulative random number generator)	495
InverseCumulativeRsg (Inverse cumulative random sequence generator)	496
IQDCurrency (Iraqi dinar)	497
IRRCurrency (Iranian rial)	498
ISKCurrency (Iceland krona)	499
Italy (Italian calendars)	500
ITLCurrency (Italian lira)	502
JamshidianSwaptionEngine (Jamshidian swaption engine)	503
JarrowRudd (Jarrow-Rudd (multiplicative) equal probabilities binomial tree)	504
Johannesburg (Johannesburg calendar)	505
JointCalendar (Joint calendar)	506
JPYCurrency (Japanese yen)	507
JPYLibor (JPY Libor index, also known as TIBOR)	508

JumpDiffusionEngine (Jump-diffusion engine for vanilla options)	509
JuQuadraticApproximationEngine	510
KnuthUniformRng (Uniform random number generator)	511
KronrodIntegral (Integral of a 1-dimensional function using the Gauss-Kronrod method)	512
KRWCurrency (South-Korean won)	513
KWDCurrency (Kuwaiti dinar)	514
Lattice (Lattice-method base class)	515
Lattice2D (Two-dimensional lattice)	517
LatticeShortRateModelEngine (Engine for a short-rate model specialized on a lattice)	518
LazyObject (Framework for calculation on demand and result caching)	519
LeastSquareFunction (Cost function for least-square problems)	521
LeastSquareProblem (Base class for least square problem)	522
LecuyerUniformRng (Uniform random number generator)	523
LeisenReimer (Leisen & Reimer tree: multiplicative approach)	524
LexicographicalView (Lexicographical 2-D view of a contiguous set of data)	525
Linear (Linear interpolation traits)	527
LinearInterpolation (Linear interpolation between discrete points)	528
LineSearch (Base class for line search)	529
Link (Relinkable access to a shared pointer)	531
LocalConstantVol (Constant local volatility, no time-strike dependence)	533
LocalVolCurve (Local volatility curve derived from a Black curve)	534
LocalVolSurface (Local volatility surface derived from a Black vol surface)	536
LocalVolTermStructure (Local-volatility term structure)	538
LogLinear (Log-linear interpolation traits)	540
LogLinearInterpolation	541
lowest_category_iterator (Most generic of two given iterator categories)	542
LTLCurrency (Lithuanian litas)	543
LUFCurrency (Luxembourg franc)	544
LVLCurrency (Latvian lat)	545
MakeSchedule (Helper class)	546
Matrix (Matrix used in linear algebra)	547
MatrixFormatter (Format matrices for output)	551
MCAmericanBasketEngine (Least-square Monte Carlo engine)	552
MCBarrierEngine (Pricing engine for barrier options using Monte Carlo simulation)	553
MCBasketEngine (Pricing engine for basket options using Monte Carlo simulation)	555
McCliquetOption (Simple example of Monte Carlo pricer)	557
MCDigitalEngine (Pricing engine for digital options using Monte Carlo simulation)	558
MCDiscreteArithmeticAPEngine (Monte Carlo pricing engine for discrete arithmetic average price Asian)	560
McDiscreteArithmeticAPO (Example of Monte Carlo pricer using a control variate)	562
McDiscreteArithmeticASO (Example of Monte Carlo pricer using a control variate)	563
MCDiscreteAveragingAsianEngine (Pricing engine for discrete average Asians using Monte Carlo simulation)	564
MCDiscreteGeometricAPEngine (Monte Carlo pricing engine for discrete geometric average price Asian)	566
MCEuropeanEngine (European option pricing engine using Monte Carlo simulation)	567
McEverest (Everest-type option pricer)	568
McHimalaya (Himalayan-type option pricer)	569
McMaxBasket (Simple example of multi-factor Monte Carlo pricer)	570
McPagoda (Roofed Asian option)	571
McPerformanceOption (Performance option computed using Monte Carlo simulation)	572
McPricer (Base class for Monte Carlo pricers)	573
McSimulation (Base class for Monte Carlo engines)	574
MCVanillaEngine (Pricing engine for vanilla options using Monte Carlo simulation)	576

MersenneTwisterUniformRng (Uniform random number generator)	578
Merton76Process (Merton-76 jump-diffusion process)	579
MixedScheme (Mixed (explicit/implicit) scheme for finite difference methods)	581
Money (Amount of cash)	583
MoneyFormatter (Format money for output)	585
MonotonicCubicSpline (Cubic spline with monotonicity constraint)	586
MonteCarloModel (General purpose Monte Carlo model for path samples)	587
MoreGreeks (More additional option results)	588
MoroInverseCumulativeNormal (Moro Inverse cumulative normal distribution class)	589
MTLCurrency (Maltese lira)	590
MultiAssetOption (Base class for options on multiple assets)	591
MultiAssetOption::arguments (Arguments for multi-asset option calculation)	593
MultiAssetOption::results (Results from multi-asset option calculation)	594
MultiCubicSpline	595
MultiPath (Correlated multiple asset paths)	596
MultiPathGenerator (Generates a multipath from a random number generator)	597
MXNCurrency (Mexican peso)	598
NaturalCubicSpline (Cubic spline with null second derivative at end points)	599
NaturalMonotonicCubicSpline (Natural cubic spline with monotonicity constraint)	600
NeumannBC (Neumann boundary condition (i.e., constant derivative))	601
Newton (Newton 1-D solver)	602
NewtonSafe (Safe Newton 1-D solver)	603
NLGCurrency (Dutch guilder)	604
NoConstraint (No constraint)	605
NOKCurrency (Norwegian krone)	606
NonLinearLeastSquare (Non-linear least-square method)	607
NormalDistribution (Normal distribution function)	608
NPRCurrency (Nepal rupee)	609
Null (Template class providing a null value for a given type)	610
NullCalendar (Calendar for reproducing theoretical calculations)	611
NullParameter (Parameter which is always zero $a(t) = 0$)	612
NumericalMethod (Numerical method (tree, finite-differences) base class)	613
NZDCurrency (New Zealand dollar)	615
Observable (Object that notifies its changes to a set of observables)	616
Observer (Object that gets notified when a given observable changes)	617
OneAssetOption (Base class for options on a single asset)	619
OneAssetOption::arguments (Arguments for single-asset option calculation)	622
OneAssetOption::results (Results from single-asset option calculation)	623
OneAssetStrikedOption (Base class for options on a single asset with striked payoff)	624
OneDayCounter (1/1 day count convention)	626
OneFactorAffineModel (Single-factor affine base class)	627
OneFactorModel (Single-factor short-rate model abstract class)	628
OneFactorModel::ShortRateDynamics (Base class describing the short-rate dynamics)	629
OneFactorModel::ShortRateTree (Recombining trinomial tree discretizing the state variable)	630
OneFactorOperator (Interest-rate single factor model differential operator)	631
OptimizationMethod (Abstract class for constrained optimization method)	632
Option (Base option class)	634
Option::arguments	635
OptionTypeFormatter (Format option type for output)	636
OrnsteinUhlenbeckProcess (Ornstein-Uhlenbeck process class)	637
Oslo (Oslo calendar)	639
Parameter (Base class for model arguments)	640
ParameterImpl (Base class for model parameter implementation)	641

ParCoupon (coupon at par on a term structure)	642
Path	644
PathGenerator (Generates random paths using a sequence generator)	645
PathPricer (Base class for path pricers)	646
Payoff (Base class for option payoffs)	647
PercentageStrikePayoff (Payoff with strike expressed as percentage)	648
Period (Time period described by a number of a given time unit)	649
PiecewiseConstantParameter (Piecewise-constant parameter)	650
PiecewiseFlatForward (Piecewise flat forward term structure)	651
PKRCurrency (Pakistani rupee)	654
PlainVanillaPayoff (Plain-vanilla payoff)	655
PLNCurrency (Polish zloty)	656
PoissonDistribution (Normal distribution function)	657
PositiveConstraint (Constraint imposing positivity to all arguments)	658
PricingEngine (Interface for pricing engines)	659
PrimeNumbers (Prime numbers calculator)	660
Problem (Constrained optimization problem)	661
processing_iterator (Iterator mapping a unary function to an underlying sequence)	662
PTECurrency (Portuguese escudo)	664
QuantoEngine (Quanto engine base class)	665
QuantoForwardVanillaOption (Quanto version of a forward vanilla option)	667
QuantoOptionArguments (Arguments for quanto option calculation)	669
QuantoOptionResults (Results from quanto option calculation)	670
QuantoTermStructure (Quanto term structure)	671
QuantoVanillaOption (Quanto version of a vanilla option)	673
Quote (Purely virtual base class for market observables)	675
RamdomizedLDS (Randomized (random shift) low-discrepancy sequence)	676
RandomSequenceGenerator (Random sequence generator based on a pseudo-random number generator)	678
RateFormatter (Formats rates for output)	679
RateHelper (Base class for rate helpers)	680
Results (Base class for generic result groups)	682
Ridder (Ridder 1-D solver)	683
Riyadh (Riyadh calendar)	684
ROLCurrency (Romanian leu)	685
Rounding (Basic rounding class)	686
SalvagingAlgorithm (Algorithm used for matricial pseudo square root)	688
Sample (Weighted sample)	689
SARCurrency (Saudi riyal)	690
Schedule (Payment schedule)	691
Secant (Secant 1-D solver)	692
SeedGenerator (Random seed generator)	693
SegmentIntegral (Integral of a one-dimensional function)	694
SEKCurrency (Swedish krona)	695
Seoul (Seoul calendar)	696
SequenceFormatter (Formats numeric sequences for output)	697
SequenceStatistics (Statistics analysis of N-dimensional (sequence) data)	698
Settings (Global repository for run-time library settings)	700
SGDCurrency (Singapore dollar)	702
Short (Short indexed coupon)	703
Short< ParCoupon > (Short coupon at par on a term structure)	704
ShortRateModel (Abstract short-rate model class)	705
ShoutCondition (Shout option condition)	707
SimpleCashFlow (Predetermined cash flow)	708

SimpleDayCounter (Simple day counter for reproducing theoretical calculations)	709
SimpleQuote (Market element returning a stored value)	710
SimpleSwap (Simple fixed-rate vs Libor swap)	711
SimpleSwap::arguments (Arguments for simple swap calculation)	713
SimpleSwap::results (Results from simple swap calculation)	714
Simplex (Multi-dimensional simplex class)	715
SimpsonIntegral (Integral of a one-dimensional function)	716
Singapore (Singapore calendar)	717
SingleAssetOption (Black-Scholes-Merton option)	718
Singleton (Basic support for the singleton pattern)	720
SITCurrency (Slovenian tolar)	721
SizeFormatter (Formats unsigned integers for output)	722
SKKCurrency (Slovak koruna)	723
SobolRsg (Sobol low-discrepancy sequence generator)	724
Solver1D (Base class for 1-D solvers)	726
SquareRootProcess (Square-root process class)	728
StatsHolder (Helper class for precomputed distributions)	729
SteepestDescent (Multi-dimensional steepest-descent class)	730
step_iterator (Iterator advancing in constant steps)	731
StepCondition (Condition to be applied at every time step)	732
stepping_iterator (Iterator advancing in constant steps)	733
StochasticProcess (Stochastic process class)	735
StochasticProcess::discretization (Discretization of a stochastic process over a given time interval)	737
Stock (Simple stock class)	738
Stockholm (Stockholm calendar)	739
StrikedTypePayoff (Intermediate class for payoffs based on a fixed strike)	740
StringFormatter (Formats strings as lower- or uppercase)	741
StulzEngine (Pricing engine for 2D European Baskets)	742
SuperSharePayoff (Binary supershare payoff)	743
SVD (Singular value decomposition)	744
Swap (Interest rate swap)	745
SwapRateHelper (Swap rate)	747
Swaption (Swaption class)	749
Swaption::arguments (Arguments for swaption calculation)	750
Swaption::results (Results from swaption calculation)	751
SwaptionVolatilityMatrix (At-the-money swaption-volatility matrix)	752
SwaptionVolatilityStructure (Swaption-volatility structure)	753
Sydney (Sydney calendar (New South Wales, Australia))	755
SymmetricSchurDecomposition (Symmetric threshold Jacobi algorithm)	756
Taiwan (Taiwan calendar)	757
TARGET (TARGET calendar)	758
TermStructureConsistentModel (Term-structure consistent model class)	759
TermStructureFittingParameter (Deterministic time-dependent parameter used for yield-curve fitting)	760
THBCurrency (Thai baht)	761
Thirty360 (30/360 day count convention)	762
Tian (Tian tree: third moment matching, multiplicative approach)	763
TimeBasket (Distribution over a number of dates)	764
TimeGrid (Time grid class)	765
Tokyo (Tokyo calendar)	766
Toronto (Toronto calendar)	768
TrapezoidIntegral (Integral of a one-dimensional function)	769
Tree (Tree approximating a single-factor diffusion)	771

TreeCapFloorEngine (Numerical lattice engine for cap/floors)	772
TreeSwaptionEngine (Numerical lattice engine for swaptions)	773
TridiagonalOperator (Base implementation for tridiagonal operator)	774
TridiagonalOperator::TimeSetter (Encapsulation of time-setting logic)	776
Trigeorgis (Trigeorgis (additive equal jumps) binomial tree)	777
TrinomialBranching (Branching scheme for a trinomial node)	778
TrinomialTree (Recombining trinomial tree class)	779
TRLCurrency (Turkish lira)	780
TTDCurrency (Trinidad & Tobago dollar)	781
TWDCurrency (Taiwan dollar)	782
TwoFactorModel (Abstract base-class for two-factor models)	783
TwoFactorModel::ShortRateDynamics (Class describing the dynamics of the two state variables)	784
TwoFactorModel::ShortRateTree (Recombining two-dimensional tree discretizing the state variable)	785
TypePayoff (Intermediate class for call/put/straddle payoffs)	786
UnitedKingdom (United Kingdom calendars)	787
UnitedStates (United States calendars)	789
UpFrontIndexedCoupon (up front indexed coupon class)	792
UpRounding (Up-rounding)	793
USDCurrency (U.S. dollar)	794
USDLibor (USD Libor index)	795
Value (Pricing results)	796
VanillaOption (Vanilla option (no discrete dividends, no barriers) on a single asset)	797
VanillaOption::engine (Vanilla option engine base class)	798
Vasicek (Vasicek model class)	799
Vasicek::Dynamics (Short-rate dynamics in the Vasicek model)	800
VEBCurrency (Venezuelan bolivar)	801
Visitor (Visitor for a specific class)	802
VolatilityFormatter (Formats volatilities for output)	803
Warsaw (Warsaw calendar)	804
WeekdayFormatter (Formats weekday for output)	805
Wellington (Wellington calendar)	806
Xibor (Base class for libor indexes)	807
XiborManager (Global repository for libor histories)	809
YieldTermStructure (Interest-rate term structure)	810
ZARCurrency (South-African rand)	817
ZARLibor (ZAR Libor index, also known as JIBAR)	818
ZeroCurve (Term structure based on linear interpolation of zero yields)	819
ZeroSpreadedTermStructure (Term structure with an added spread on the zero yield rate)	821
ZeroYieldStructure (Zero-yield term structure)	823
Zurich (Zurich calendar)	825

Chapter 5

QuantLib File Index

5.1 QuantLib File List

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

ql/argsandresults.hpp (Base classes for generic arguments and results)	827
ql/basetermstructure.hpp (Base class for term structures)	829
ql/basicdataformatters.hpp (Classes used to format basic types for output)	830
ql/calendar.hpp (calendar class)	831
ql/capvolstructures.hpp (Cap/Floor volatility structures)	857
ql/cashflow.hpp (Base class for cash flows)	859
ql/currency.hpp (Known currencies)	886
ql/dataformatters.hpp (Classes used to format data for output)	888
ql/dataparsers.hpp (Classes used to parse data for input)	889
ql/date.hpp (Date- and time-related classes, typedefs and enumerations)	890
ql/daycounter.hpp (Day counter class)	892
ql/discretizedasset.hpp (Discretized asset classes)	900
ql/disposable.hpp (Generic disposable object with move semantics)	901
ql/errors.hpp (Classes and functions for error handling)	902
ql/exchangerate.hpp (Exchange rate between two currencies)	904
ql/exercise.hpp (Option exercise classes and payoff function)	905
ql/grid.hpp (Grid classes with useful constructors for trees and finite diffs)	925
ql/history.hpp (History class)	926
ql/index.hpp (Purely virtual base class for indexes)	927
ql/instrument.hpp (Abstract instrument class)	939
ql/interestrate.hpp (Instrument rate class)	962
ql/money.hpp (Cash amount in a given currency)	1017
ql/null.hpp (Null values)	1029
ql/numericalmethod.hpp (Numerical method class)	1030
ql/option.hpp (Base option class)	1042
ql/payoff.hpp (Option payoff classes)	1050
ql/pricingengine.hpp (Base class for pricing engines)	1073
ql/qldefines.hpp (Global definitions and compiler switches)	1119
ql/quote.hpp (Purely virtual base class for market observables)	1121
ql/relinkablehandle.hpp (Globally accessible relinkable pointer)	1139
ql/schedule.hpp (Date schedule)	1140
ql/settings.hpp (Global repository for run-time library settings)	1141
ql/solver1d.hpp (Abstract 1-D solver class)	1155

ql/ stochasticprocess.hpp (Stochastic processes)	1163
ql/ swaptionvolstructure.hpp (Swaption volatility structure)	1164
ql/ termstructure.hpp (Term structure)	1165
ql/ types.hpp (Custom types)	1182
ql/ voltermstructure.hpp (Volatility term structures)	1200
ql/Calendars/ beijing.hpp (Beijing calendar)	832
ql/Calendars/ budapest.hpp (Budapest calendar)	833
ql/Calendars/ copenhagen.hpp (Copenhagen calendar)	834
ql/Calendars/ germany.hpp (German calendars)	835
ql/Calendars/ helsinki.hpp (Helsinki calendar)	836
ql/Calendars/ hongkong.hpp (Hong Kong calendar)	837
ql/Calendars/ italy.hpp (Italian calendars)	838
ql/Calendars/ johannesburg.hpp (Johannesburg calendar)	839
ql/Calendars/ jointcalendar.hpp (Joint calendar)	840
ql/Calendars/ nullcalendar.hpp (Calendar for reproducing theoretical calculations)	841
ql/Calendars/ oslo.hpp (Oslo calendar)	842
ql/Calendars/ riyadh.hpp (Riyadh calendar)	843
ql/Calendars/ seoul.hpp (South Korea calendar)	844
ql/Calendars/ singapore.hpp (Singapore calendar)	845
ql/Calendars/ stockholm.hpp (Stockholm calendar)	846
ql/Calendars/ sydney.hpp (Sydney calendar)	847
ql/Calendars/ taiwan.hpp (Taiwan calendar)	848
ql/Calendars/ target.hpp (TARGET calendar)	849
ql/Calendars/ tokyo.hpp (Tokyo calendar)	850
ql/Calendars/ toronto.hpp (Toronto calendar)	851
ql/Calendars/ unitedkingdom.hpp (UK calendars)	852
ql/Calendars/ unitedstates.hpp (US calendars)	853
ql/Calendars/ warsaw.hpp (Warsaw calendar)	854
ql/Calendars/ wellington.hpp (Wellington calendar)	855
ql/Calendars/ zurich.hpp (Zurich calendar)	856
ql/CashFlows/ basispointsensitivity.hpp (Basis point sensitivity calculator)	860
ql/CashFlows/ cashflowvectors.hpp (Cash flow vector builders)	861
ql/CashFlows/ coupon.hpp (Coupon accruing over a fixed period)	863
ql/CashFlows/ fixedratecoupon.hpp (Coupon paying a fixed annual rate)	864
ql/CashFlows/ floatingratecoupon.hpp (Coupon paying a variable rate)	865
ql/CashFlows/ inarrearindexedcoupon.hpp (In-arrear floating-rate coupon)	866
ql/CashFlows/ indexcashflowvectors.hpp (Index Cash flow vector builders)	867
ql/CashFlows/ indexedcoupon.hpp (Indexed coupon)	869
ql/CashFlows/ parcoupon.hpp (Coupon at par on a term structure)	870
ql/CashFlows/ shortfloatingcoupon.hpp (Short (or long) coupon at par on a term structure)	871
ql/CashFlows/ shortindexedcoupon.hpp (Short (or long) indexed coupon)	872
ql/CashFlows/ simplecashflow.hpp (Predetermined cash flow)	873
ql/CashFlows/ timebasket.hpp	874
ql/CashFlows/ upfrontindexedcoupon.hpp (Up front indexed coupon)	875
ql/Currencies/ africa.hpp (African currencies)	876
ql/Currencies/ america.hpp (American currencies)	877
ql/Currencies/ asia.hpp (Asian currencies)	879
ql/Currencies/ europe.hpp (European currencies)	881
ql/Currencies/ exchangeratemanager.hpp (Exchange-rate repository)	884
ql/Currencies/ oceania.hpp (Oceanian currencies)	885
ql/DayCounters/ actual360.hpp (Act/360 day counter)	893
ql/DayCounters/ actual365.hpp (Act/365 day counter)	894
ql/DayCounters/ actual365fixed.hpp (Actual/365 (Fixed) day counter)	895
ql/DayCounters/ actualactual.hpp (Act/act day counters)	896

ql/DayCounters/ one.hpp (1/1 day counter)	897
ql/DayCounters/ simplifiedaycounter.hpp (Simple day counter for reproducing theoretical calculations)	898
ql/DayCounters/ thirty360.hpp (30/360 day counters)	899
ql/FiniteDifferences/ americancondition.hpp (American option exercise condition)	906
ql/FiniteDifferences/ boundarycondition.hpp (Boundary conditions for differential operators)	907
ql/FiniteDifferences/ bsmoperator.hpp (Differential operator for Black-Scholes-Merton equation)	908
ql/FiniteDifferences/ cranknicolson.hpp (Crank-Nicolson scheme for finite difference methods)	909
ql/FiniteDifferences/ dminus.hpp (D_- matricial representation)	910
ql/FiniteDifferences/ dplus.hpp (D_+ matricial representation)	911
ql/FiniteDifferences/ dplusdminus.hpp (D_+D_- matricial representation)	912
ql/FiniteDifferences/ dzero.hpp (D_0 matricial representation)	913
ql/FiniteDifferences/ explicit euler.hpp (Explicit Euler scheme for finite difference methods)	914
ql/FiniteDifferences/ fdtypedefs.hpp (Default choices for template instantiations)	915
ql/FiniteDifferences/ finitedifferencemodel.hpp (Generic finite difference model)	916
ql/FiniteDifferences/ implicit euler.hpp (Implicit Euler scheme for finite difference methods)	917
ql/FiniteDifferences/ mixedscheme.hpp (Mixed (explicit/implicit) scheme for finite difference methods)	918
ql/FiniteDifferences/ onefactoroperator.hpp (General differential operator for one-factor interest rate models)	919
ql/FiniteDifferences/ shoutcondition.hpp (Shout option exercise condition)	920
ql/FiniteDifferences/ stepcondition.hpp (Conditions to be applied at every time step) . .	921
ql/FiniteDifferences/ tridiagonaloperator.hpp (Tridiagonal operator)	922
ql/FiniteDifferences/ valueatcenter.hpp (Compute value, first, and second derivatives at grid center)	923
ql/Indexes/ audlibor.hpp (AUD Libor index (check settlement days))	928
ql/Indexes/ cadlibor.hpp (CAD Libor index (Also known as CDOR))	929
ql/Indexes/ chflibor.hpp (CHF Libor index (Also known as ZIBOR))	930
ql/Indexes/ euribor.hpp (Euribor index)	931
ql/Indexes/ gbplibor.hpp (GBP Libor index)	932
ql/Indexes/ indexmanager.hpp (Global repository for past index fixings)	933
ql/Indexes/ jpylibor.hpp (JPY Libor index (Also known as TIBOR, check settlement days)) .	934
ql/Indexes/ usdlibor.hpp (USD Libor index)	935
ql/Indexes/ xibor.hpp (Base class for libor indexes)	936
ql/Indexes/ xibormanager.hpp (Global repository for Xibor histories)	937
ql/Indexes/ zarlibor.hpp (ZAR Libor index (also known as JIBAR))	938
ql/Indexes/ asianoption.hpp (Asian option on a single asset)	940
ql/Instruments/ barrieroption.hpp (Barrier option on a single asset)	941
ql/Instruments/ basketoption.hpp (Basket option on a number of assets)	942
ql/Instruments/ bond.hpp (Concrete bond class)	943
ql/Instruments/ capfloor.hpp (Cap and Floor class)	944
ql/Instruments/ cliquetoption.hpp (Cliquet option)	945
ql/Instruments/ dividendvanillaoption.hpp (Vanilla option on a single asset with discrete dividends)	946
ql/Instruments/ europeanoption.hpp (European option on a single asset)	947
ql/Instruments/ fixedcouponbond.hpp (Fixed-coupon bond)	948
ql/Instruments/ forwardvanillaoption.hpp (Forward version of a vanilla option)	949
ql/Instruments/ multiassetoption.hpp (Option on multiple assets)	950
ql/Instruments/ oneassetoption.hpp (Option on a single asset)	951

ql/Instruments/ oneassetstrikedoption.hpp (Option on a single asset with striked payoff)	952
ql/Instruments/ payoffs.hpp (Payoffs for various options)	953
ql/Instruments/ quantoforwardvanillaoption.hpp (Quanto version of a forward vanilla option)	955
ql/Instruments/ quantovanillaoption.hpp (Quanto version of a vanilla option)	956
ql/Instruments/ simpleswap.hpp (Simple fixed-rate vs Libor swap)	957
ql/Instruments/ stock.hpp (Concrete stock class)	958
ql/Instruments/ swap.hpp (Interest rate swap)	959
ql/Instruments/ swaption.hpp (Swaption class)	960
ql/Instruments/ vanillaoption.hpp (Vanilla option on a single asset)	961
ql/Lattices/ binomialtree.hpp (Binomial tree class)	963
ql/Lattices/ bsmlattice.hpp (Binomial trees under the BSM model)	965
ql/Lattices/ lattice.hpp (Lattice method class)	966
ql/Lattices/ lattice2d.hpp (Two-dimensional lattice class)	967
ql/Lattices/ tree.hpp (Tree class)	968
ql/Lattices/ trinomialtree.hpp (Trinomial tree class)	969
ql/Math/ array.hpp (1-D array used in linear algebra)	970
ql/Math/ beta.hpp (Beta and beta incomplete functions)	971
ql/Math/ bicubicsplineinterpolation.hpp (Bicubic spline interpolation between discrete points)	972
ql/Math/ bilinearinterpolation.hpp (Bilinear interpolation between discrete points)	973
ql/Math/ binomialdistribution.hpp (Binomial distribution)	974
ql/Math/ bivariatenormaldistribution.hpp (Bivariate cumulative normal distribution)	976
ql/Math/ chisquaredistribution.hpp (Chi-square (central and non-central) distributions)	977
ql/Math/ choleskydecomposition.hpp (Cholesky decomposition)	978
ql/Math/ comparison.hpp (Floating-point comparisons)	979
ql/Math/ cubicspline.hpp (Cubic spline interpolation between discrete points)	980
ql/Math/ discrepancystatistics.hpp (Statistic tool for sequences with discrepancy calculation)	981
ql/Math/ errorfunction.hpp (Error function)	982
ql/Math/ extrapolation.hpp (Class-wide extrapolation settings)	983
ql/Math/ factorial.hpp (Factorial numbers calculator)	984
ql/Math/ functional.hpp (Functionals and combinators not included in the STL)	985
ql/Math/ gammadistribution.hpp (Gamma distribution)	986
ql/Math/ gaussianstatistics.hpp (Statistics tool for gaussian-assumption risk measures)	987
ql/Math/ generalstatistics.hpp (Statistics tool)	988
ql/Math/ incompletegamma.hpp (Incomplete Gamma function)	989
ql/Math/ incrementalstatistics.hpp (Statistics tool based on incremental accumulation)	990
ql/Math/ interpolation.hpp (Base class for 1-D interpolations)	991
ql/Math/ interpolation2D.hpp (Abstract base classes for 2-D interpolations)	992
ql/Math/ interpolationtraits.hpp (Traits classes for interpolation algorithms)	993
ql/Math/ kronrodintegral.hpp (Integral of a 1-dimensional function using the Gauss-Kronrod method)	994
ql/Math/ lexicographicalview.hpp (Lexicographical 2-D view of a contiguous set of data)	995
ql/Math/ linearinterpolation.hpp (Linear interpolation between discrete points)	996
ql/Math/ loglinearinterpolation.hpp (Log-linear interpolation between discrete points)	997
ql/Math/ matrix.hpp (Matrix used in linear algebra)	998
ql/Math/ multicubicspline.hpp (N-dimensional cubic spline interpolation between discrete points)	999
ql/Math/ normaldistribution.hpp (Normal, cumulative and inverse cumulative distributions)	1001
ql/Math/ poissondistribution.hpp (Poisson distribution)	1002
ql/Math/ primenumbers.hpp (Prime numbers calculator)	1003
ql/Math/ pseudosqrt.hpp (Pseudo square root of a real symmetric matrix)	1004

ql/Math/riskstatistics.hpp (Empirical-distribution risk measures)	1005
ql/Math/rounding.hpp (Rounding implementation)	1007
ql/Math/segmentintegral.hpp (Integral of a one-dimensional function)	1008
ql/Math/sequencestatistics.hpp (Statistics tools for sequence (vector, list, array) samples)	1009
ql/Math/simpsonintegral.hpp (Integral of a one-dimensional function)	1011
ql/Math/statistics.hpp (Statistics tool with risk measures)	1012
ql/Math/svd.hpp (Singular value decomposition)	1013
ql/Math/symmetriceigenvalues.hpp (Eigenvalues / eigenvectors of a real symmetric matrix)	1014
ql/Math/symmetricschurdecomposition.hpp (Eigenvalues / eigenvectors of a real symmetric matrix)	1015
ql/Math/trapezoidintegral.hpp (Integral of a one-dimensional function)	1016
ql/MonteCarlo/brownianbridge.hpp (Browian bridge)	1018
ql/MonteCarlo/getcovariance.hpp (Covariance matrix calculation)	1019
ql/MonteCarlo/mctraits.hpp (Monte Carlo policies)	1020
ql/MonteCarlo/mctypedefs.hpp (Default choices for template instantiations)	1021
ql/MonteCarlo/montecarlomodel.hpp (General purpose Monte Carlo model)	1022
ql/MonteCarlo/multipath.hpp (Correlated multiple asset paths)	1023
ql/MonteCarlo/multipathgenerator.hpp (Generates a multi path from a random-array generator)	1024
ql/MonteCarlo/path.hpp (Single factor random walk)	1025
ql/MonteCarlo/pathgenerator.hpp (Generates random paths using a sequence generator)	1026
ql/MonteCarlo/pathpricer.hpp (Base class for single-path pricers)	1027
ql/MonteCarlo/sample.hpp (Weighted sample)	1028
ql/Optimization/armijo.hpp (Armijo line-search class)	1031
ql/Optimization/conjugategradient.hpp (Conjugate gradient optimization method)	1032
ql/Optimization/constraint.hpp (Abstract constraint class)	1033
ql/Optimization/costfunction.hpp (Optimization cost function class)	1034
ql/Optimization/criteria.hpp (Optimization criteria class)	1035
ql/Optimization/leastsquare.hpp (Least square cost function)	1036
ql/Optimization/linesearch.hpp (Line search abstract class)	1037
ql/Optimization/method.hpp (Abstract optimization method class)	1038
ql/Optimization/problem.hpp (Abstract optimization class)	1039
ql/Optimization/simplex.hpp (Simplex optimization method)	1040
ql/Optimization/steepestdescent.hpp (Steepest descent optimization method)	1041
ql/Patterns/bridge.hpp (Bridge pattern (a.k.a. handle-body idiom))	1043
ql/Patterns/composite.hpp (Composite pattern)	1044
ql/Patterns/curiouslyrecurring.hpp (Curiously recurring template pattern)	1045
ql/Patterns/lazyobject.hpp (Framework for calculation on demand and result caching)	1046
ql/Patterns/observable.hpp (Observer/observable pattern)	1047
ql/Patterns/singleton.hpp (Basic support for the singleton pattern)	1048
ql/Patterns/visitor.hpp (Degenerate base class for the Acyclic Visitor pattern)	1049
ql/Pricers/discretegeometricapo.hpp (Discrete Geometric Average Price Option)	1051
ql/Pricers/discretegeometricaso.hpp (Discrete Geometric Average Strike Option)	1052
ql/Pricers/fdamericanoption.hpp (American option)	1053
ql/Pricers/fdbermudanoption.hpp (Finite-difference evaluation of Bermudan option)	1054
ql/Pricers/fdbsmoption.hpp (Common code for numerical option evaluation)	1055
ql/Pricers/fddividendamericanoption.hpp (American option with discrete deterministic dividends)	1056
ql/Pricers/fddividendoption.hpp (Base class for option with dividends)	1057
ql/Pricers/fddividendshoutoption.hpp (Base class for shout option with dividends)	1058
ql/Pricers/fdeuropean.hpp (Example of European option calculated using finite differences)	1059

ql/Pricers/ fdmultiperiodoption.hpp (Base class for option with events happening at different periods)	1060
ql/Pricers/ fdshoutoption.hpp (Shout option)	1061
ql/Pricers/ fdstepconditionoption.hpp (Option requiring additional code to be executed at each time step)	1062
ql/Pricers/ mccliquetoption.hpp (Cliquet option priced with Monte Carlo simulation) . .	1063
ql/Pricers/ mcdiscretearithmeticapo.hpp (Discrete Arithmetic Average Price Option) . .	1064
ql/Pricers/ mcdiscretearithmeticaso.hpp (Discrete Arithmetic Average Strike Option) . .	1065
ql/Pricers/ mceverest.hpp (Everest-type option pricer)	1066
ql/Pricers/ mchimalaya.hpp (Himalayan-type option pricer)	1067
ql/Pricers/ mcmaxbasket.hpp (Max Basket Monte Carlo pricer)	1068
ql/Pricers/ mcpagoda.hpp (Roofed multi asset Asian option)	1069
ql/Pricers/ mcperformanceoption.hpp (Performance option priced with Monte Carlo simulation)	1070
ql/Pricers/ mcpricer.hpp (Base class for Monte Carlo pricers)	1071
ql/Pricers/ singleassetoption.hpp (Common code for option evaluation)	1072
ql/PricingEngines/ americanpayoffatexpiry.hpp (Analytical formulae for american exercise with payoff at expiry)	1074
ql/PricingEngines/ americanpayoffathit.hpp (Analytical formulae for american exercise with payoff at hit)	1075
ql/PricingEngines/ blackformula.hpp (Black formula)	1086
ql/PricingEngines/ blackmodel.hpp (Abstract class for Black-type models (market models))	1087
ql/PricingEngines/ genericmodelengine.hpp (Generic option engine based on a model) .	1097
ql/PricingEngines/ latticeshortratemodelengine.hpp (Engine for a short-rate model specialized on a lattice)	1098
ql/PricingEngines/ mcsimulation.hpp (Framework for Monte Carlo engines)	1099
ql/PricingEngines/Asian/ analytic_cont_geom_av_price.hpp (Analytic engine for continuous geometric average price Asian)	1076
ql/PricingEngines/Asian/ analytic_discr_geom_av_price.hpp (Analytic engine for discrete geometric average price Asian)	1077
ql/PricingEngines/Asian/ mc_discr_arith_av_price.hpp (Monte Carlo engine for discrete arithmetic average price Asian)	1078
ql/PricingEngines/Asian/ mc_discr_geom_av_price.hpp (Monte Carlo engine for discrete geometric average price Asian)	1079
ql/PricingEngines/Asian/ mcdiscreteasianengine.hpp (Monte Carlo pricing engine for discrete average Asians)	1080
ql/PricingEngines/Barrier/ analyticbarrierengine.hpp (Analytic barrier option engines) .	1081
ql/PricingEngines/Barrier/ mcbarrierengine.hpp (Monte Carlo barrier option engines) .	1082
ql/PricingEngines/Basket/ mcamericanbasketengine.hpp (Least-square Monte Carlo engines)	1083
ql/PricingEngines/Basket/ mcbasketengine.hpp (European basket MC Engine)	1084
ql/PricingEngines/Basket/ stulzengine.hpp (2D European Basket formulae, due to Stulz (1982))	1085
ql/PricingEngines/CapFloor/ analyticcapfloorengine.hpp (Analytic engine for caps/floors)	1088
ql/PricingEngines/CapFloor/ blackcapfloorengine.hpp (Black-formula cap/floor engine)	1089
ql/PricingEngines/CapFloor/ discretizedcapfloor.hpp (Discretized cap/floor)	1090
ql/PricingEngines/CapFloor/ treecapfloorengine.hpp (Numerical lattice engine for cap/floors)	1091
ql/PricingEngines/Cliquet/ analyticcliquetengine.hpp (Analytic Cliquet engine)	1092
ql/PricingEngines/Cliquet/ analyticperformanceengine.hpp (Analytic performance engine)	1093
ql/PricingEngines/Cliquet/ mccliquetengine.hpp (Monte Carlo Cliquet option engine) .	1094

ql/PricingEngines/Forward/ forwardengine.hpp (Forward (strike-resetting) option engine)	1095
ql/PricingEngines/Forward/ forwardperformanceengine.hpp (Forward (strike-resetting) performance option engines)	1096
ql/PricingEngines/Quanto/ quantoengine.hpp (Quanto option engine)	1100
ql/PricingEngines/Swaption/ blackswaptionengine.hpp (Black-formula swaption engine)	1101
ql/PricingEngines/Swaption/ discretizedswaption.hpp (Discretized swaption class)	1102
ql/PricingEngines/Swaption/ g2swaptionengine.hpp (Swaption pricing engine for two-factor additive Gaussian Model G2++)	1103
ql/PricingEngines/Swaption/ jamshidianswaptionengine.hpp (Swaption engine using Jamshidian's decomposition)	1104
ql/PricingEngines/Swaption/ treeswaptionengine.hpp (Numerical lattice engine for swaptions)	1105
ql/PricingEngines/Vanilla/ analyticdigitalamericanengine.hpp (Analytic digital American option engine)	1106
ql/PricingEngines/Vanilla/ analyticdividendeuropeanengine.hpp (Analytic discrete-dividend European engine)	1107
ql/PricingEngines/Vanilla/ analyticeuropeanengine.hpp (Analytic European engine)	1108
ql/PricingEngines/Vanilla/ baroneadesiwhaleyengine.hpp (Barone-Adesi and Whaley approximation engine)	1109
ql/PricingEngines/Vanilla/ binomialengine.hpp (Binomial option engine)	1110
ql/PricingEngines/Vanilla/ bjerk sundstenslandengine.hpp (Bjerk sund and Stensland approximation engine)	1111
ql/PricingEngines/Vanilla/ discretizedvanillaoption.hpp (Discretized vanilla option)	1112
ql/PricingEngines/Vanilla/ integralengine.hpp (Integral option engine)	1113
ql/PricingEngines/Vanilla/ jumpdiffusionengine.hpp (Jump diffusion (Merton 1976) engine)	1114
ql/PricingEngines/Vanilla/ juquadraticengine.hpp (Ju quadratic (1999) approximation engine)	1115
ql/PricingEngines/Vanilla/ mcdigitalengine.hpp (Digital option Monte Carlo engine)	1116
ql/PricingEngines/Vanilla/ mceuropeanengine.hpp (Monte Carlo European option engine)	1117
ql/PricingEngines/Vanilla/ mcvanillaengine.hpp (Monte Carlo vanilla option engine)	1118
ql/RandomNumbers/ boxmullergaussianrng.hpp (Box-Muller Gaussian random-number generator)	1122
ql/RandomNumbers/ centrallimitgaussianrng.hpp (Central limit Gaussian random-number generator)	1123
ql/RandomNumbers/ faurersg.hpp (Faure low-discrepancy sequence generator)	1124
ql/RandomNumbers/ haltonrng.hpp (Halton low-discrepancy sequence generator)	1125
ql/RandomNumbers/ inversecumgaussianrng.hpp (Inverse cumulative Gaussian random-number generator)	1126
ql/RandomNumbers/ inversecumgaussianrngs.hpp (Inverse cumulative Gaussian random sequence generator)	1127
ql/RandomNumbers/ inversecumulativerng.hpp (Inverse cumulative Gaussian random-number generator)	1128
ql/RandomNumbers/ inversecumulativersg.hpp (Inverse cumulative random sequence generator)	1129
ql/RandomNumbers/ knuthuniformrng.hpp (Knuth uniform random number generator)	1130
ql/RandomNumbers/ lecuyeruniformrng.hpp (L'Ecuyer uniform random number generator)	1131
ql/RandomNumbers/ mt19937uniformrng.hpp (Mersenne Twister uniform random number generator)	1132
ql/RandomNumbers/ randomizedlds.hpp (Randomized low-discrepancy sequence)	1133

ql/RandomNumbers/ randomsequencegenerator.hpp (Random sequence generator based on a pseudo-random number generator)	1134
ql/RandomNumbers/ rngtraits.hpp (Random-number generation policies)	1135
ql/RandomNumbers/ seedgenerator.hpp (Random seed generator)	1137
ql/RandomNumbers/ sobolrsg.hpp (Sobol low-discrepancy sequence generator)	1138
ql/ShortRateModels/ calibrationhelper.hpp (Calibration helper class)	1142
ql/ShortRateModels/ model.hpp (Abstract interest rate model class)	1145
ql/ShortRateModels/ onefactormodel.hpp (Abstract one-factor interest rate model class)	1146
ql/ShortRateModels/ parameter.hpp (Model parameter classes)	1152
ql/ShortRateModels/ twofactormodel.hpp (Abstract two-factor interest rate model class)	1153
ql/ShortRateModels/CalibrationHelpers/ caphelper.hpp (CapHelper calibration helper)	1143
ql/ShortRateModels/CalibrationHelpers/ swaptionhelper.hpp (Swaption calibration helper)	1144
ql/ShortRateModels/OneFactorModels/ blackkarasinski.hpp (Black-Karasinski model)	1147
ql/ShortRateModels/OneFactorModels/ coxingersollross.hpp (Cox-Ingersoll-Ross model)	1148
ql/ShortRateModels/OneFactorModels/ extendedcoxingersollross.hpp (Extended Cox-Ingersoll-Ross model)	1149
ql/ShortRateModels/OneFactorModels/ hullwhite.hpp (Hull & White (HW) model)	1150
ql/ShortRateModels/OneFactorModels/ vasicek.hpp (Vasicek model class)	1151
ql/ShortRateModels/TwoFactorModels/ g2.hpp (Two-factor additive Gaussian Model G2++)	1154
ql/Solvers1D/ bisection.hpp (Bisection 1-D solver)	1156
ql/Solvers1D/ brent.hpp (Brent 1-D solver)	1157
ql/Solvers1D/ falseposition.hpp (False-position 1-D solver)	1158
ql/Solvers1D/ newton.hpp (Newton 1-D solver)	1159
ql/Solvers1D/ newtonsafe.hpp (Safe (bracketed) Newton 1-D solver)	1160
ql/Solvers1D/ ridder.hpp (Ridder 1-D solver)	1161
ql/Solvers1D/ secant.hpp (Secant 1-D solver)	1162
ql/TermStructures/ affinetermstructure.hpp (Affine term structure)	1166
ql/TermStructures/ compoundforward.hpp (Compounded forward term structure)	1167
ql/TermStructures/ discountcurve.hpp (Pre-bootstrapped discount factor structure)	1168
ql/TermStructures/ discountstructure.hpp (Discount-based yield term structure)	1169
ql/TermStructures/ drifttermstructure.hpp (Drift term structure)	1170
ql/TermStructures/ extendeddiscountcurve.hpp (Discount factor structure with detailed compound-forward calculation)	1171
ql/TermStructures/ flatforward.hpp (Flat forward rate term structure)	1172
ql/TermStructures/ forwardspreadedtermstructure.hpp (Forward-spreaded term structure)	1173
ql/TermStructures/ forwardstructure.hpp (Forward-based yield term structure)	1174
ql/TermStructures/ impliedtermstructure.hpp (Implied term structure)	1175
ql/TermStructures/ piecewiseflatforward.hpp (Piecewise flat forward term structure)	1176
ql/TermStructures/ quantotermstructure.hpp (Quanto term structure)	1177
ql/TermStructures/ ratehelpers.hpp (Rate helpers base class)	1178
ql/TermStructures/ zerocurve.hpp (Pre-bootstrapped zero curve structure)	1179
ql/TermStructures/ zerospreadedtermstructure.hpp (Zero spreaded term structure)	1180
ql/TermStructures/ zeroyieldstructure.hpp (Zero-yield based term structure)	1181
ql/Utilities/ combiningiterator.hpp (Iterator mapping a function to a set of underlying sequences)	1184
ql/Utilities/ couplingiterator.hpp (Iterator mapping a function to a pair of underlying sequences)	1185
ql/Utilities/ filteringiterator.hpp (Iterator filtering undesired data)	1186
ql/Utilities/ iteratorcategories.hpp (Lowest common denominator between two iterator categories)	1187

ql/Utilities/ processingiterator.hpp (Iterator mapping a unary function to an underlying sequence)	1188
ql/Utilities/ steppingiterator.hpp (Iterator advancing in constant steps)	1189
ql/Volatilities/ blackconstantvol.hpp (Black constant volatility, no time dependence, no strike dependence)	1190
ql/Volatilities/ blackvariancecurve.hpp (Black volatility curve modelled as variance curve)	1191
ql/Volatilities/ blackvariancesurface.hpp (Black volatility surface modelled as variance surface)	1192
ql/Volatilities/ capflatvolvector.hpp (Cap/floor at-the-money flat volatility vector)	1193
ql/Volatilities/ capletconstantvol.hpp (Constant caplet volatility)	1194
ql/Volatilities/ impliedvoltermstructure.hpp (Implied Black Vol Term Structure)	1195
ql/Volatilities/ localconstantvol.hpp (Local constant volatility, no time dependence, no asset dependence)	1196
ql/Volatilities/ localvolcurve.hpp (Local volatility curve derived from a Black curve) . .	1197
ql/Volatilities/ localvolsurface.hpp (Local volatility surface derived from a Black vol surface)	1198
ql/Volatilities/ swaptionvolmatrix.hpp (Swaption at-the-money volatility matrix)	1199

Chapter 6

QuantLib Module Documentation

6.1 Numeric types

6.1.1 Detailed Description

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

Typedefs

- typedef QL_INTEGER [QuantLib::Integer](#)
integer number
- typedef QL_BIG_INTEGER [QuantLib::BigInteger](#)
large integer number
- typedef unsigned QL_INTEGER [QuantLib::Natural](#)
positive integer
- typedef QL_REAL [QuantLib::Real](#)
real number
- typedef [Real](#) [QuantLib::Decimal](#)
decimal number
- typedef QL_SIZE_T [QuantLib::Size](#)
size of a container
- typedef [Real](#) [QuantLib::Time](#)
continuous quantity with 1-year units
- typedef [Real](#) [QuantLib::DiscountFactor](#)
discount factor between dates
- typedef [Real](#) [QuantLib::Rate](#)
interest rates

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

6.2 Currencies and FX rates

Classes

- class [ZARCurrency](#)
South-African rand.
- class [CADCurrency](#)
Canadian dollar.
- class [USDCurrency](#)
U.S. dollar.
- class [JPYCurrency](#)
Japanese yen.
- class [CHFCurrency](#)
Swiss franc.
- class [EURCurrency](#)
European Euro.
- class [GBPCurrency](#)
British pound sterling.
- class [DEMCurrency](#)
Deutsche mark.
- class [ITLCurrency](#)
Italian lira.
- class [AUDCurrency](#)
Australian dollar.

Enumerations

- enum [QuantLib::CurrencyTag](#) {
 [QuantLib::ARS](#), [QuantLib::ATS](#), [QuantLib::AUD](#), [QuantLib::BDT](#),
 [QuantLib::BEF](#), [QuantLib::BGL](#), [QuantLib::BRL](#), [QuantLib::BYB](#),
 [QuantLib::CAD](#), [QuantLib::CHF](#), [QuantLib::CLP](#), [QuantLib::CNY](#),
 [QuantLib::COP](#), [QuantLib::CYP](#), [QuantLib::CZK](#), [QuantLib::DEM](#),
 [QuantLib::DKK](#), [QuantLib::EEK](#), [QuantLib::EUR](#), [QuantLib::GBP](#),
 [QuantLib::GRD](#), [QuantLib::HKD](#), [QuantLib::HUF](#), [QuantLib::ILS](#),
 [QuantLib::INR](#), [QuantLib::IQD](#), [QuantLib::IRR](#), [QuantLib::ISK](#),
 [QuantLib::ITL](#), [QuantLib::JPY](#), [QuantLib::KRW](#), [QuantLib::KWD](#),
 [QuantLib::LTL](#), [QuantLib::LVL](#), [QuantLib::MTL](#), [QuantLib::MXP](#),

```
QuantLib::NOK, QuantLib::NPR, QuantLib::NZD, QuantLib::PKR,  
QuantLib::PLN, QuantLib::ROL, QuantLib::SAR, QuantLib::SEK,  
QuantLib::SGD, QuantLib::SIT, QuantLib::SKK, QuantLib::THB,  
QuantLib::TRL, QuantLib::TTD, QuantLib::TWD, QuantLib::USD,  
QuantLib::VEB, QuantLib::ZAR }
```

Tags for known currencies.

6.2.1 Enumeration Type Documentation

6.2.1.1 enum [CurrencyTag](#)

Tags for known currencies.

Deprecated

use [Currency](#) instead

Enumeration values:

ARS Argentinian Peso.
ATS Austrian Schillings.
AUD Australian Dollar.
BDT Bangladesh Taka.
BEF Belgian Franc.
BGL Bulgarian Lev.
BRL Brazilian Real.
BYB Belarusian Ruble.
CAD Canadian Dollar.
CHF Swiss Franc.
CLP Chilean Peso.
CNY Chinese Yuan.
COP Colombian Peso.
CYP Cyprus Pound.
CZK Czech Koruna.
DEM German Mark.
DKK Danish Krone.
EEK Estonian Kroon.
EUR Euro.
GBP British Pound.
GRD Greek Drachma.
HKD Hong Kong Dollar.
HUF Hungarian Forint.
ILS Israeli Shekel.
INR Indian Rupee.

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

6.3 Date and time calculations

6.3.1 Detailed Description

The concrete class [QuantLib::Date](#) implements the concept of date. Its functionalities include:

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

Modules

- [Calendars](#)
- [Day counters](#)

Classes

- class [Calendar](#)
calendar class
- class [Period](#)
Time period described by a number of a given time unit.
- class [Date](#)
Concrete date class.
- class [DayCounter](#)
day counter class

Typedefs

- typedef [Integer QuantLib::Day](#)
Day number.
- typedef [Integer QuantLib::Year](#)
Year number.

Enumerations

- enum `QuantLib::BusinessDayConvention` {
`QuantLib::Unadjusted`, `QuantLib::Preceding`, `QuantLib::ModifiedPreceding`, `QuantLib::Following`,
`QuantLib::ModifiedFollowing`, `QuantLib::MonthEndReference` }
Business Day conventions.
- enum `QuantLib::Weekday` {
`Sunday` = 1, `Monday` = 2, `Tuesday` = 3, `Wednesday` = 4,
`Thursday` = 5, `Friday` = 6, `Saturday` = 7 }
- enum `QuantLib::Month` {
`January` = 1, `February` = 2, `March` = 3, `April` = 4,
`May` = 5, `June` = 6, `July` = 7, `August` = 8,
`September` = 9, `October` = 10, `November` = 11, `December` = 12 }
Month names.
- enum `QuantLib::IMMMonth` { `H` = 3, `M` = 6, `U` = 9, `Z` = 12 }
Main cycle of the International Money Market (a.k.a. IMM) Months.
- enum `QuantLib::Frequency` {
`QuantLib::NoFrequency` = -1, `QuantLib::Once` = 0, `QuantLib::Annual` = 1, `QuantLib::Semiannual` = 2,
`QuantLib::EveryFourthMonth` = 3, `QuantLib::Quarterly` = 4, `QuantLib::Bimonthly` = 6,
`QuantLib::Monthly` = 12 }
Frequency of events.
- enum `QuantLib::TimeUnit` { `Days`, `Weeks`, `Months`, `Years` }
Units used to describe time periods.

6.3.2 Enumeration Type Documentation

6.3.2.1 enum `BusinessDayConvention`

Business Day conventions.

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

Enumeration values:

Unadjusted Do not adjust.

Preceding Choose the first business day before the given holiday.

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

Following Choose the first business day after the given holiday.

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

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

6.3.2.2 enum **Weekday**

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

6.3.2.3 enum **Frequency**

Frequency of events.

Enumeration values:

NoFrequency null frequency

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

Annual once a year

Semiannual twice a year

EveryFourthMonth every fourth month

Quarterly every third month

Bimonthly every second month

Monthly once a month

6.4 Calendars

6.4.1 Detailed Description

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

Classes

- class `Beijing`
Beijing calendar
- class `Budapest`
Budapest calendar
- class `Copenhagen`
Copenhagen calendar
- class `Germany`
German calendars.
- class `Helsinki`
Helsinki calendar
- class `HongKong`
Hong Kong calendar.
- class `Italy`
Italian calendars.
- class `Johannesburg`
Johannesburg calendar
- class `JointCalendar`
Joint calendar.
- class `NullCalendar`
Calendar for reproducing theoretical calculations.
- class `Oslo`
Oslo calendar
- class `Riyadh`
Riyadh calendar
- class `Seoul`
Seoul calendar

- class [Singapore](#)
Singapore calendar
- class [Stockholm](#)
Stockholm calendar
- class [Sydney](#)
Sydney calendar (New South Wales, Australia)
- class [Taiwan](#)
Taiwan calendar
- class [TARGET](#)
TARGET calendar
- class [Tokyo](#)
Tokyo calendar
- class [Toronto](#)
Toronto calendar
- class [UnitedKingdom](#)
United Kingdom calendars.
- class [UnitedStates](#)
United States calendars.
- class [Warsaw](#)
Warsaw calendar
- class [Wellington](#)
Wellington calendar
- class [Zurich](#)
Zurich calendar

6.5 Day counters

6.5.1 Detailed Description

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

Classes

- class [Actual360](#)
Actual/360 day count convention.
- class [Actual365Fixed](#)
Actual/365 (Fixed) day count convention.
- class [ActualActual](#)
Actual/Actual day count.
- class [OneDayCounter](#)
1/1 day count convention
- class [SimpleDayCounter](#)
Simple day counter for reproducing theoretical calculations.
- class [Thirty360](#)
30/360 day count convention

Typedefs

- typedef `Actual365Fixed` [QuantLib::Actual365](#)

6.5.2 Typedef Documentation

6.5.2.1 typedef `Actual365Fixed` [Actual365](#)

Actual/365 day count convention

As per ISDA documentation Actual/365 is the same as Actual/Actual. You should use the [Actual-Actual](#) class instead, or maybe you want to use the [Actual365Fixed](#) class.

Deprecated

use [ActualActual](#) or [Actual365Fixed](#) instead

6.6 Pricing engines

Modules

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

6.7 Asian option engines

Classes

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

6.8 Barrier option engines

Classes

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

6.9 Basket option engines

Classes

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

6.10 Cap/floor engines

Classes

- class [AnalyticCapFloorEngine](#)
Analytic engine for cap/floor.
- class [BlackCapFloorEngine](#)
Black-formula cap/floor engine.
- class [TreeCapFloorEngine](#)
Numerical lattice engine for cap/floors.

6.11 Cliquet option engines

Classes

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

6.12 Forward option engines

Classes

- class [ForwardEngine](#)
Forward engine base class.
- class [ForwardPerformanceEngine](#)
Forward performance engine.

6.13 Quanto option engines

Classes

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

6.14 Swaption engines

Classes

- class [BlackSwaptionEngine](#)
Black-formula swaption engine.
- class [G2SwaptionEngine](#)
Swaption priced by means of the Black formula
- class [JamshidianSwaptionEngine](#)
Jamshidian swaption engine.
- class [TreeSwaptionEngine](#)
Numerical lattice engine for swaptions.

6.15 Vanilla option engines

Classes

- class [AnalyticDigitalAmericanEngine](#)
- class [AnalyticDividendEuropeanEngine](#)
Analytic pricing engine for European options with discrete dividends.
- class [AnalyticEuropeanEngine](#)
Pricing engine for European vanilla options using analytical formulae.
- class [BaroneAdesiWhaleyApproximationEngine](#)
- class [BinomialVanillaEngine](#)
Pricing engine for vanilla options using binomial trees.
- class [Bjerk SundStenslandApproximationEngine](#)
- class [IntegralEngine](#)
- class [JumpDiffusionEngine](#)
Jump-diffusion engine for vanilla options.
- class [JuQuadraticApproximationEngine](#)
- class [MCDigitalEngine](#)
Pricing engine for digital options using Monte Carlo simulation.
- class [MCEuropeanEngine](#)
European option pricing engine using Monte Carlo simulation.
- class [MCVanillaEngine](#)
Pricing engine for vanilla options using Monte Carlo simulation.

6.16 Finite-differences framework

6.16.1 Detailed Description

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

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

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

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

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

6.16.2 Differential operators

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

6.16.3 Time schemes

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

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

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

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

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

hence

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

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

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

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

hence

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

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

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

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

hence

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

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

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

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

However, it must implement the following interface:

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

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

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

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

6.16.4 Step conditions

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

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

6.16.5 An example of finite difference model

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

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

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

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

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

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

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

```

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

```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

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

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

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

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

```
unsigned int timeSteps = 365;

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

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

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

Classes

- class [BoundaryCondition](#)

Abstract boundary condition class for finite difference problems.

- class [NeumannBC](#)

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

- class [DirichletBC](#)
Neumann boundary condition (i.e., constant value).
- class [BSMOperator](#)
Black-Scholes-Merton differential operator.
- class [CrankNicolson](#)
Crank-Nicolson scheme for finite difference methods.
- class [DMinus](#)
 D_- matricial representation
- class [DPlus](#)
 D_+ matricial representation
- class [DPlusDMinus](#)
 D_+D_- matricial representation
- class [DZero](#)
 D_0 matricial representation
- class [ExplicitEuler](#)
Forward Euler scheme for finite difference methods.
- class [FiniteDifferenceModel](#)
Generic finite difference model.
- class [ImplicitEuler](#)
Backward Euler scheme for finite difference methods.
- class [MixedScheme](#)
Mixed (explicit/implicit) scheme for finite difference methods.
- class [OneFactorOperator](#)
Interest-rate single factor model differential operator.
- class [StepCondition](#)
condition to be applied at every time step
- class [TridiagonalOperator](#)
Base implementation for tridiagonal operator.

6.17 Short-rate modelling framework

6.17.1 Detailed Description

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

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

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

6.17.2 Single-factor models

The Hull & White model

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

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

The Black-Karasinski model

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

No analytical tractability here.

The extended Cox-Ingersoll-Ross model

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

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

6.17.3 Calibration

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

Derived classes are `QuantLib::CapHelper` and `QuantLib::SwaptionHelper`.

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

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

where T_i is the price given by the model and M_i is the market price, is minimized. A few optimization methods are available in the `ql/Optimization` directory.

6.17.4 Two-factor models

6.17.5 Pricers

Analytical pricers

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

Using Finite Differences

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

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

any european-style instrument will follow the following PDE:

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

The adequate operator to feed a Finite Difference Model instance is defined in the `QuantLib::OneFactorOperator` class.

Using Trees

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

Classes

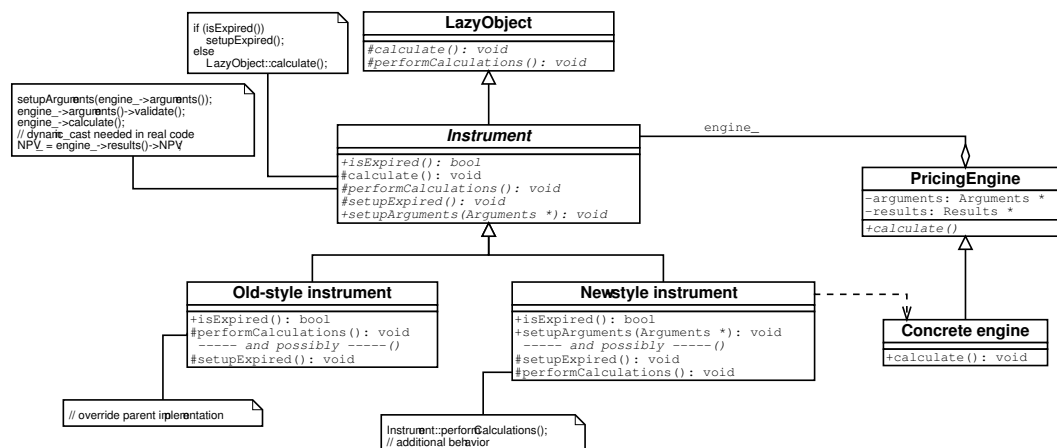
- class `AffineModel`
Affine model class.
- class `TermStructureConsistentModel`
Term-structure consistent model class.
- class `ShortRateModel`
Abstract short-rate model class.
- class `OneFactorModel`
Single-factor short-rate model abstract class.

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

6.18 Financial instruments

6.18.1 Detailed Description

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



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

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

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

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

class FooOptionArguments : public FooArguments { ... };

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

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

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

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

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

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

Classes

- class [ContinuousAveragingAsianOption](#)
Continuous-averaging Asian option.
- class [DiscreteAveragingAsianOption](#)
Discrete-averaging Asian option.
- class [BarrierOption](#)
Barrier option on a single asset.
- class [BasketOption](#)
Basket option on a number of assets.
- class [Bond](#)
Base bond class.
- class [CapFloor](#)
Base class for cap-like instruments.
- class [Cap](#)
Concrete cap class.
- class [Floor](#)
Concrete floor class.
- class [Collar](#)
Concrete collar class.
- class [CliquetOption](#)
cliquet (Ratchet) option
- class [DividendVanillaOption](#)
Single-asset vanilla option (no barriers) with discrete dividends.
- class [EuropeanOption](#)
European option on a single asset.

- class [FixedCouponBond](#)
fixed-coupon bond
- class [ForwardVanillaOption](#)
Forward version of a vanilla option.
- class [QuantoForwardVanillaOption](#)
Quanto version of a forward vanilla option.
- class [QuantoVanillaOption](#)
quanto version of a vanilla option
- class [SimpleSwap](#)
Simple fixed-rate vs Libor swap.
- class [Stock](#)
Simple stock class.
- class [Swap](#)
Interest rate swap.
- class [Swaption](#)
Swaption class
- class [VanillaOption](#)
Vanilla option (no discrete dividends, no barriers) on a single asset.

6.19 Lattice methods

6.19.1 Detailed Description

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

6.19.2 Binomial trees

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

6.19.3 Trinomial trees

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

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

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

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

and

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

where k (the index of the node at the end of the middle branch) is the index of the node which is the nearest to the expected future value, $E_{i,j} = \mathbf{E}(y(t_{i+1})|y(t_i) = y_{i,j})$ and $V_{i,j}^2 = \mathbf{Var}\{y(t_{i+1})|y(t_i) = y_{i,j}\}$.

If we suppose that the variance is only dependant on time $V_{i,j} = V_i$ and set y_{i+1} to $V_i \sqrt{3}$, we find that

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

6.19.4 Bidimensional lattices

To come...

6.19.5 The QuantLib::DiscretizedAsset class

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

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

Classes

- class [BinomialTree](#)
Binomial tree base class.
- class [EqualProbabilitiesBinomialTree](#)
Base class for equal probabilities binomial tree.
- class [EqualJumpsBinomialTree](#)
Base class for equal jumps binomial tree.
- class [JarrowRudd](#)
Jarrow-Rudd (multiplicative) equal probabilities binomial tree.
- class [CoxRossRubinstein](#)
Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.
- class [AdditiveEQPBinomialTree](#)
Additive equal probabilities binomial tree.
- class [Trigeorgis](#)
Trigeorgis (additive equal jumps) binomial tree
- class [Tian](#)
Tian tree: third moment matching, multiplicative approach
- class [LeisenReimer](#)
Leisen & Reimer tree: multiplicative approach.
- class [BlackScholesLattice](#)
Simple binomial lattice approximating the Black-Scholes model.
- class [Lattice](#)
Lattice-method base class.
- class [Lattice2D](#)

Two-dimensional lattice.

- class [Tree](#)

Tree approximating a single-factor diffusion

- class [TrinomialTree](#)

Recombining trinomial tree class.

6.20 Math tools

Math facilities of the library include:

6.20.1 Pseudo-random number and low-discrepancy sequence generators

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

6.20.2 One-dimensional solvers

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

A number of such solvers is contained in the `ql/Solvers1D` directory.

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

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

6.20.3 Optimizers

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

The simplex method

This method, implemented in [QuantLib::Simplex](#), is rather raw and requires quite a lot of computing resources, but it has the advantage that it does not need any evaluation of the cost function's gradient, and that it is quite easily implemented. First, we must choose $N+1$ starting points, given here by a starting point \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](#). At each step, we minimize (using Armijo's line search algorithm, implemented in [QuantLib::ArmijoLineSearch](#)) the function along a line defined by

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

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

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

6.21 Monte Carlo framework

6.21.1 Detailed Description

Warning: this section of the documentation is currently outdated.

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

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

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

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

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

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

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

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

6.21.2 Path generation

The Black-Scholes equation

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

where r is the risk-free rate, 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), paths followed by the logarithm of the underlying would be Brownian random walks with a constant drift v per unit time and a standard deviation $\sigma\sqrt{T}$ over a time T .

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

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

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

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

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

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

6.21.3 Pricing an instrument on a path

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

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

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

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

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

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

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

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

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

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

6.21.4 Accumulating and averaging samples

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

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

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

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

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

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

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

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

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

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

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

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

6.21.5 Examples of Monte Carlo models

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

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

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

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

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

The corresponding path pricer can be implemented as:

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

        size_t n = path.size();

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

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

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

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

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

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

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

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

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

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

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

The path generator can be instantiated as

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

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

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

The path pricer is instantiated as

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

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

The model can now be created and used as following:

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

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

model.addSamples(samples);

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

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

6.21.6 Notes

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

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

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

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

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

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

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

Classes

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

6.22 Design patterns

Classes

- class [Bridge](#)
The Bridge pattern made explicit.
- class [Composite](#)
Composite pattern.
- class [CuriouslyRecurringTemplate](#)
Support for the curiously recurring template pattern.
- class [LazyObject](#)
Framework for calculation on demand and result caching.
- class [Observable](#)
Object that notifies its changes to a set of observables.
- class [Observer](#)
Object that gets notified when a given observable changes.
- class [Singleton](#)
Basic support for the singleton pattern.
- class [AcyclicVisitor](#)
degenerate base class for the Acyclic Visitor pattern

6.23 Term structures

6.23.1 Detailed Description

The abstract class [QuantLib::YieldTermStructure](#) provides the common interface to concrete yield-rate term structure models. Among others, methods are declared which return instantaneous forward rate, discount factor, and zero rate at a given date. Adapter classes are provided which already implement part of the required methods, thus allowing the programmer to define only the non-redundant part.

Classes

- class [YieldTermStructure](#)
Interest-rate term structure.
- class [DiscountCurve](#)
Term structure based on loglinear interpolation of discount factors.
- class [DiscountStructure](#)
Discount factor term structure.
- class [FlatForward](#)
Flat interest-rate curve.
- class [ForwardSpreadedTermStructure](#)
Term structure with added spread on the instantaneous forward rate.
- class [ForwardRateStructure](#)
Forward rate term structure.
- class [ImpliedTermStructure](#)
Implied term structure at a given date in the future.
- class [PiecewiseFlatForward](#)
Piecewise flat forward term structure.
- class [ZeroCurve](#)
Term structure based on linear interpolation of zero yields.
- class [ZeroSpreadedTermStructure](#)
Term structure with an added spread on the zero yield rate.
- class [ZeroYieldStructure](#)
Zero-yield term structure.

6.24 Utilities

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

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

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

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

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

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

[QuantLib::coupling_iterator](#) allows us to do the same thing without allocating the extra vector: what we do is simply:

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

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

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

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

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

6.25 QuantLib macros

6.25.1 Detailed Description

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

Modules

- [Generic macros](#)
- [Math functions](#)
- [Numeric limits](#)
- [Time functions](#)
- [String functions](#)
- [Character functions](#)
- [Min and max functions](#)
- [Template capabilities](#)
- [Iterator support](#)

Defines

- `#define QL_VERSION "0.3.8"`
version string
- `#define QL_HEX_VERSION 0x000308f0`
version hexadecimal number
- `#define QL_LIB_VERSION "0_3_8"`
version string for output lib name

6.26 Generic macros

6.26.1 Detailed Description

Miscellaneous macros for compiler idiosyncrasies not fitting other categories.

Defines

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

6.26.2 Define Documentation

6.26.2.1 `#define QL_DUMMY_RETURN(x)`

Is a dummy return statement required?

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

```
if (condition)
    return validResult;
else
    QL_FAIL("whatever the reason");
```

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

6.26.2.2 `#define QL_IO_INIT`

I/O initialization.

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

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

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

6.27 Math functions

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

6.28 Numeric limits

6.28.1 Detailed Description

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

Defines

- `#define QL_MIN_INTEGER ((std::numeric_limits<QL_INTEGER>::min)())`
- `#define QL_MAX_INTEGER ((std::numeric_limits<QL_INTEGER>::max)())`
- `#define QL_MIN_REAL -((std::numeric_limits<QL_REAL>::max)())`
- `#define QL_MIN_POSITIVE_REAL ((std::numeric_limits<QL_REAL>::min)())`
- `#define QL_MAX_REAL ((std::numeric_limits<QL_REAL>::max)())`
- `#define QL_EPSILON ((std::numeric_limits<QL_REAL>::epsilon)())`
- `#define QL_NULL_INTEGER ((std::numeric_limits<int>::max)())`
- `#define QL_NULL_REAL ((std::numeric_limits<float>::max)())`

6.28.2 Define Documentation

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

Defines the value of the largest representable negative integer value

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

Defines the value of the largest representable integer value

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

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

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

Defines the value of the smallest representable positive double value

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

Defines the value of the largest representable floating-point value

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

Defines the machine precision for operations over doubles

6.29 Time functions

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.

6.30 String functions

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.

6.31 Character functions

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.

6.32 Min and max functions

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.

6.33 Template capabilities

6.33.1 Detailed Description

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

Defines

- `#define QL_TYPENAME typename`
Blame Microsoft for this one...

6.33.2 Define Documentation

6.33.2.1 `#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.

6.34 Iterator support

6.34.1 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_SPECIALIZE_ITERATOR_TRAITS(T)`
- `#define QL_REVERSE_ITERATOR(iterator, type) std::reverse_iterator< iterator >`
Blame Microsoft for this one...
- `#define QL_FULL_ITERATOR_SUPPORT`

6.34.2 Define Documentation

6.34.2.1 `#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.

Deprecated

no longer needed for the Boost iterator library

6.34.2.2 `#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.

Deprecated

use `boost::reverse_iterator` instead

6.34.2.3 `#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.

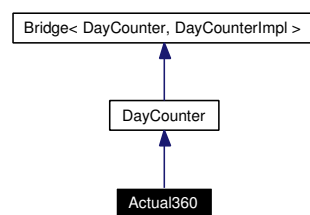
Chapter 7

QuantLib Class Documentation

7.1 Actual360 Class Reference

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

Inheritance diagram for Actual360:



7.1.1 Detailed Description

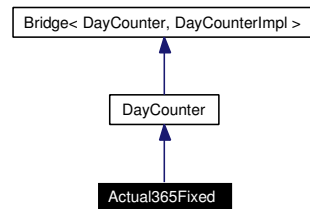
Actual/360 day count convention.

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

7.2 Actual365Fixed Class Reference

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

Inheritance diagram for Actual365Fixed:



7.2.1 Detailed Description

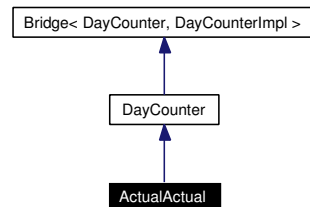
Actual/365 (Fixed) day count convention.

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

7.3 ActualActual Class Reference

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

Inheritance diagram for ActualActual:



7.3.1 Detailed Description

Actual/Actual day count.

The day count can be calculated according to:

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

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

Warning:

this is the same as "Actual/365", while "Actual/365 (Fixed)" is a different daycount

Tests

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

Public Types

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

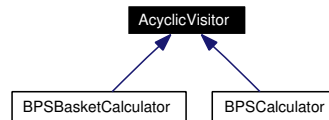
Public Member Functions

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

7.4 AcyclicVisitor Class Reference

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

Inheritance diagram for AcyclicVisitor:



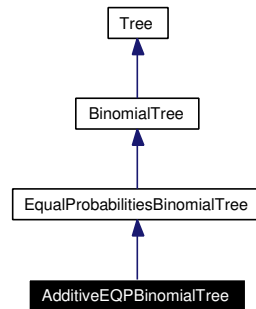
7.4.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

7.5 AdditiveEQPBinoialTree Class Reference

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

Inheritance diagram for AdditiveEQPBinoialTree:



7.5.1 Detailed Description

Additive equal probabilities binomial tree.

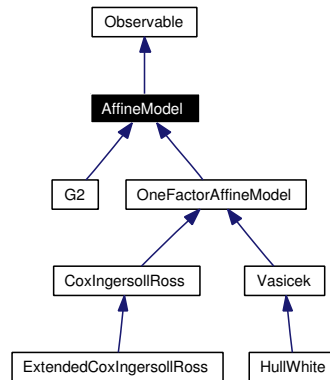
Public Member Functions

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

7.6 AffineModel Class Reference

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

Inheritance diagram for AffineModel:



7.6.1 Detailed Description

Affine model class.

Base class for analytically tractable models.

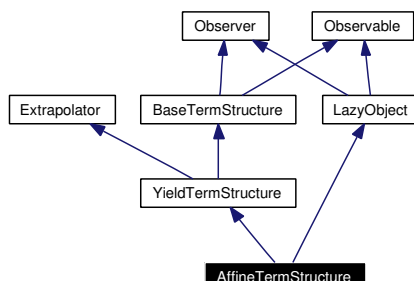
Public Member Functions

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

7.7 AffineTermStructure Class Reference

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

Inheritance diagram for AffineTermStructure:



7.7.1 Detailed Description

Term-structure implied by an affine model.

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

Public Member Functions

- **AffineTermStructure** (const [Date](#) &todayDate, const [Date](#) &referenceDate, const boost::shared_ptr< [AffineModel](#) > &model, const [DayCounter](#) &dayCounter)
constructor using a fixed model
- **AffineTermStructure** (const [Date](#) &todayDate, const [Date](#) &referenceDate, const boost::shared_ptr< [AffineModel](#) > &model, const std::vector< boost::shared_ptr< [RateHelper](#) > > &, const boost::shared_ptr< [OptimizationMethod](#) > &, const [DayCounter](#) &dayCounter)
constructor using a model that has to be calibrated
- **AffineTermStructure** (const [Date](#) &referenceDate, const boost::shared_ptr< [AffineModel](#) > &model, const [DayCounter](#) &dayCounter)
constructor using a fixed model
- **AffineTermStructure** (const [Date](#) &referenceDate, const boost::shared_ptr< [AffineModel](#) > &model, const std::vector< boost::shared_ptr< [RateHelper](#) > > &, const boost::shared_ptr< [OptimizationMethod](#) > &, const [DayCounter](#) &dayCounter)
constructor using a model that has to be calibrated
- **AffineTermStructure** ([Integer](#) settlementDays, const [Calendar](#) &calendar, const boost::shared_ptr< [AffineModel](#) > &model, const [DayCounter](#) &dayCounter)
constructor using a fixed model

- [AffineTermStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &calendar, const boost::shared_ptr< [AffineModel](#) > &model, const std::vector< boost::shared_ptr< [RateHelper](#) > > &, const boost::shared_ptr< [OptimizationMethod](#) > &, const [DayCounter](#) &dayCounter)
constructor using a model that has to be calibrated
- [DayCounter](#) dayCounter () const
the day counter used for date/time conversion
- [Date](#) maxDate () const
the latest date for which the curve can return rates
- void [update](#) ()

Protected Member Functions

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

7.7.2 Constructor & Destructor Documentation

- 7.7.2.1 [AffineTermStructure](#) (const [Date](#) & todaysDate, const [Date](#) & referenceDate, const boost::shared_ptr< [AffineModel](#) > & model, const [DayCounter](#) & dayCounter)

constructor using a fixed model

Deprecated

use the constructor without today's date.

- 7.7.2.2 [AffineTermStructure](#) (const [Date](#) & todaysDate, const [Date](#) & referenceDate, const boost::shared_ptr< [AffineModel](#) > & model, const std::vector< boost::shared_ptr< [RateHelper](#) > > &, const boost::shared_ptr< [OptimizationMethod](#) > &, const [DayCounter](#) & dayCounter)

constructor using a model that has to be calibrated

Deprecated

use the constructor without today's date.

7.7.3 Member Function Documentation

- 7.7.3.1 void [update](#) () [virtual]

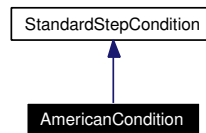
This method must be implemented in derived classes. An instance of [Observer](#) does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [LazyObject](#).

7.8 AmericanCondition Class Reference

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

Inheritance diagram for AmericanCondition:



7.8.1 Detailed Description

American exercise condition.

Todo

unify the intrinsicValues/Payoff thing

Public Member Functions

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

7.8.2 Member Function Documentation

7.8.2.1 void **applyTo** (boost::shared_ptr< [DiscretizedAsset](#) > asset) const [virtual]

Deprecated

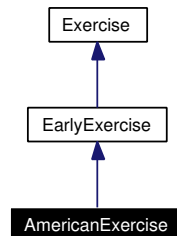
use `adjustValues()` on the asset itself

Implements [StepCondition](#).

7.9 AmericanExercise Class Reference

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

Inheritance diagram for AmericanExercise:



7.9.1 Detailed Description

American exercise.

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

Todo

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

Public Member Functions

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

7.10 AmericanPayoffAtExpiry Class Reference

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

7.10.1 Detailed Description

Analytic formula for American exercise payoff at-expiry options

Todo

calculate greeks

Public Member Functions

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

7.11 AmericanPayoffAtHit Class Reference

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

7.11.1 Detailed Description

Analytic formula for American exercise payoff at-hit options

Todo

calculate greeks

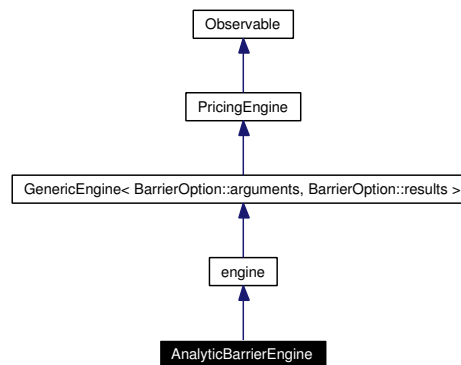
Public Member Functions

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

7.12 AnalyticBarrierEngine Class Reference

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

Inheritance diagram for AnalyticBarrierEngine:



7.12.1 Detailed Description

Pricing engine for barrier options using analytical formulae.

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

Tests

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

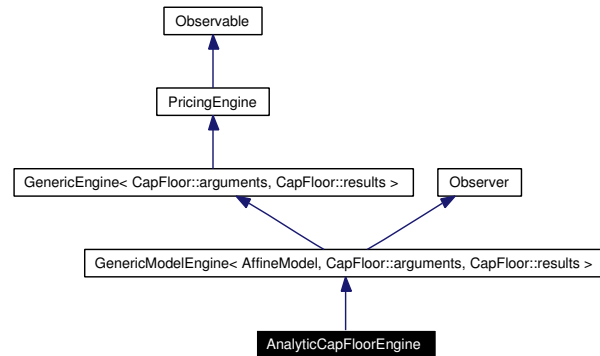
Public Member Functions

- void **calculate** () const

7.13 AnalyticCapFloorEngine Class Reference

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

Inheritance diagram for AnalyticCapFloorEngine:



7.13.1 Detailed Description

Analytic engine for cap/floor.

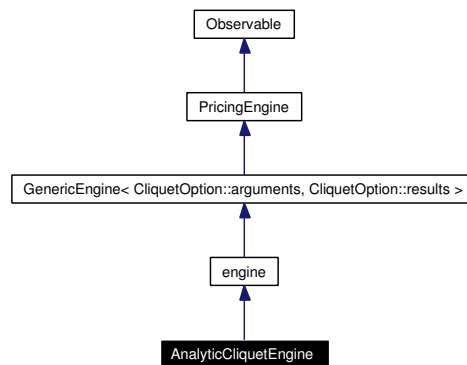
Public Member Functions

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

7.14 AnalyticCliquetEngine Class Reference

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

Inheritance diagram for AnalyticCliquetEngine:



7.14.1 Detailed Description

Pricing engine for Cliquet options using analytical formulae.

Tests

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

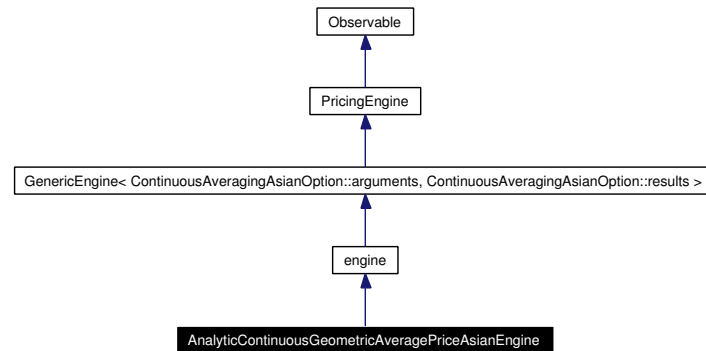
Public Member Functions

- void **calculate** () const

7.15 AnalyticContinuousGeometricAveragePriceAsianEngine Class Reference

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

Inheritance diagram for AnalyticContinuousGeometricAveragePriceAsianEngine:



7.15.1 Detailed Description

Pricing engine for European continuous geometric average price Asian.

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

Tests

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

Todo

handle seasoned options

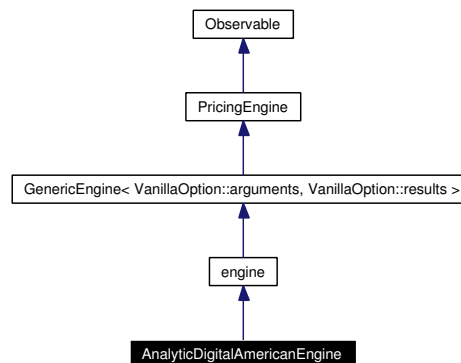
Public Member Functions

- void **calculate** () const

7.16 AnalyticDigitalAmericanEngine Class Reference

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

Inheritance diagram for AnalyticDigitalAmericanEngine:



7.16.1 Detailed Description

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

Todo

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

Tests

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

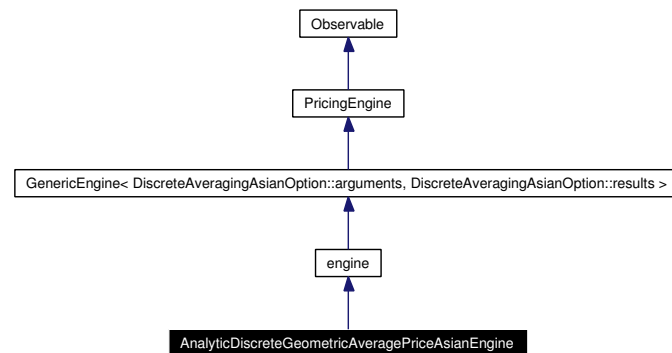
Public Member Functions

- void **calculate** () const

7.17 AnalyticDiscreteGeometricAveragePriceAsianEngine Class Reference

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

Inheritance diagram for AnalyticDiscreteGeometricAveragePriceAsianEngine:



7.17.1 Detailed Description

Pricing engine for European discrete geometric average price Asian.

This class implements a discrete geometric average price Asian option, with European exercise. The formula is from "Asian Option", E. Levy (1997) in "Exotic Options: The State of the Art", edited by L. Clewlow, C. Strickland, pag 65-97

Todo

implement correct theta, rho, dividend-rho, and vega calculation

Tests

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

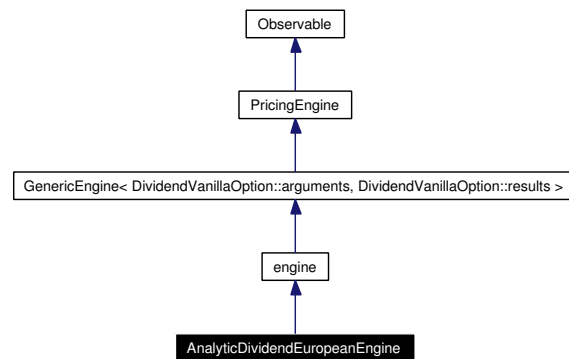
Public Member Functions

- void **calculate** () const

7.18 AnalyticDividendEuropeanEngine Class Reference

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

Inheritance diagram for AnalyticDividendEuropeanEngine:



7.18.1 Detailed Description

Analytic pricing engine for European options with discrete dividends.

Tests

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

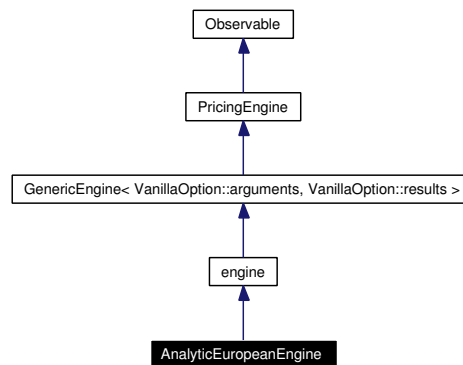
Public Member Functions

- void **calculate** () const

7.19 AnalyticEuropeanEngine Class Reference

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

Inheritance diagram for AnalyticEuropeanEngine:



7.19.1 Detailed Description

Pricing engine for European vanilla options using analytical formulae.

Tests

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing results available in literature.
- c) the correctness of the returned greeks is tested by reproducing numerical derivatives.
- d) the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- e) the implied-volatility calculation is tested by checking that it does not modify the option.
- f) the correctness of the returned value in case of cash-or-nothing digital payoff is tested by reproducing results available in literature.
- g) the correctness of the returned value in case of asset-or-nothing digital payoff is tested by reproducing results available in literature.
- h) the correctness of the returned value in case of gap digital payoff is tested by reproducing results available in literature.
- i) the correctness of the returned greeks in case of cash-or-nothing digital payoff is tested by reproducing numerical derivatives.

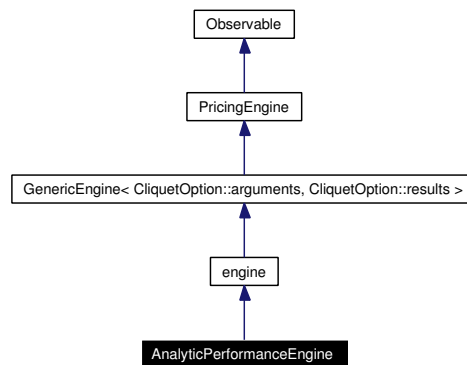
Public Member Functions

- void **calculate** () const

7.20 AnalyticPerformanceEngine Class Reference

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

Inheritance diagram for AnalyticPerformanceEngine:



7.20.1 Detailed Description

Pricing engine for performance options using analytical formulae.

Tests

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

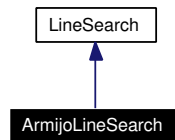
Public Member Functions

- void **calculate** () const

7.22 ArmijoLineSearch Class Reference

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

Inheritance diagram for ArmijoLineSearch:



7.22.1 Detailed Description

Armijo line search.

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

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

and

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

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

Public Member Functions

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

7.23 Array Class Reference

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

7.23.1 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 [Real](#) * **iterator**
- typedef const [Real](#) * **const_iterator**
- typedef boost::reverse_iterator< iterator > **reverse_iterator**
- typedef boost::reverse_iterator< const_iterator > **const_reverse_iterator**

Public Member Functions

Constructors, destructor, and assignment

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

Vector algebra

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

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

Precondition:

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

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

Element access

- [Real operator\[\]](#) (Size) const
read-only
- [Real & operator\[\]](#) (Size)
read-write

Inspectors

- [Size size](#) () const
dimension of the array

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](#) &)
- [Real DotProduct](#) (const [Array](#) &, const [Array](#) &)
- const [Disposable](#)< [Array](#) > [operator+](#) (const [Array](#) &v)
- const [Disposable](#)< [Array](#) > [operator-](#) (const [Array](#) &v)
- const [Disposable](#)< [Array](#) > [operator+](#) (const [Array](#) &, const [Array](#) &)
- const [Disposable](#)< [Array](#) > [operator+](#) (const [Array](#) &, [Real](#))
- const [Disposable](#)< [Array](#) > [operator+](#) ([Real](#), const [Array](#) &)
- const [Disposable](#)< [Array](#) > [operator-](#) (const [Array](#) &, const [Array](#) &)
- const [Disposable](#)< [Array](#) > [operator-](#) (const [Array](#) &, [Real](#))
- const [Disposable](#)< [Array](#) > [operator-](#) ([Real](#), const [Array](#) &)
- const [Disposable](#)< [Array](#) > [operator*](#) (const [Array](#) &, const [Array](#) &)
- const [Disposable](#)< [Array](#) > [operator*](#) (const [Array](#) &, [Real](#))
- const [Disposable](#)< [Array](#) > [operator*](#) ([Real](#), const [Array](#) &)
- const [Disposable](#)< [Array](#) > [operator/](#) (const [Array](#) &, const [Array](#) &)
- const [Disposable](#)< [Array](#) > [operator/](#) (const [Array](#) &, [Real](#))
- const [Disposable](#)< [Array](#) > [operator/](#) ([Real](#), const [Array](#) &)
- const [Disposable](#)< [Array](#) > [Abs](#) (const [Array](#) &)
- const [Disposable](#)< [Array](#) > [Sqrt](#) (const [Array](#) &)
- const [Disposable](#)< [Array](#) > [Log](#) (const [Array](#) &)
- const [Disposable](#)< [Array](#) > [Exp](#) (const [Array](#) &)

7.23.2 Friends And Related Function Documentation

7.23.2.1 `std::ostream & operator<< (std::ostream &, const Array &)` [related]

Deprecated

send to the stream the output of [ArrayFormatter](#)

7.24 ArrayFormatter Class Reference

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

7.24.1 Detailed Description

format arrays for output

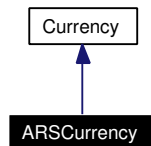
Static Public Member Functions

- `std::string toString` (const [Array](#) &a, [Integer](#) precision=6, [Integer](#) digits=0, [Size](#) elements-PerRow=QL_MAX_INTEGER)

7.25 ARSCurrency Class Reference

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

Inheritance diagram for ARSCurrency:



7.25.1 Detailed Description

Argentinian peso.

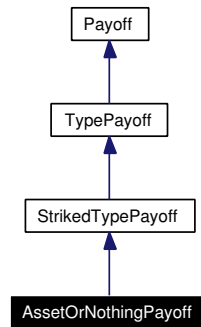
The ISO three-letter code is ARS; the numeric code is 32. It is divided in 100 centavos.

ingroup currencies

7.26 AssetOrNothingPayoff Class Reference

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

Inheritance diagram for AssetOrNothingPayoff:



7.26.1 Detailed Description

Binary asset-or-nothing payoff.

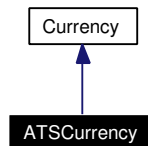
Public Member Functions

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

7.27 ATSCurrency Class Reference

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

Inheritance diagram for ATSCurrency:



7.27.1 Detailed Description

Austrian shilling.

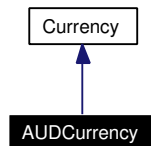
The ISO three-letter code was ATS; the numeric code was 40. It was divided in 100 groschen.

ingroup currencies

7.28 AUDCurrency Class Reference

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

Inheritance diagram for AUDCurrency:



7.28.1 Detailed Description

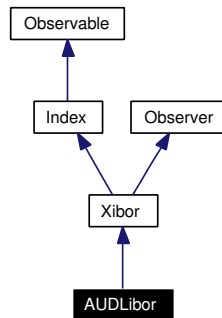
Australian dollar.

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

7.29 AUDLibor Class Reference

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

Inheritance diagram for AUDLibor:



7.29.1 Detailed Description

AUD Libor index, also known as SIBOR

Todo

check settlement days

Public Member Functions

- **AUDLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual365Fixed](#)())

7.30 Average Struct Reference

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

7.30.1 Detailed Description

placeholder for enumerated averaging types

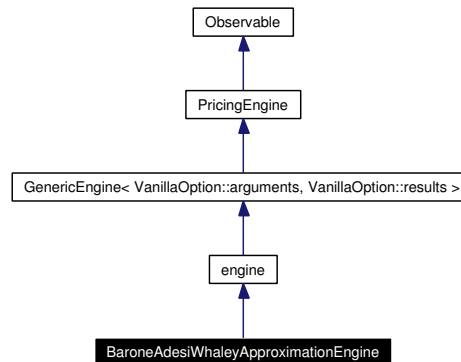
Public Types

- enum Type { Arithmetic, Geometric }

7.31 BaroneAdesiWhaleyApproximationEngine Class Reference

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

Inheritance diagram for BaroneAdesiWhaleyApproximationEngine:



7.31.1 Detailed Description

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

Tests

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

Public Member Functions

- `void calculate () const`

Static Public Member Functions

- `Real criticalPrice` (const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, [DiscountFactor](#) riskFreeDiscount, [DiscountFactor](#) dividendDiscount, [Real](#) variance, [Real](#) tolerance=1e-6)

7.32 Barrier Struct Reference

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

7.32.1 Detailed Description

Placeholder for enumerated barrier types.

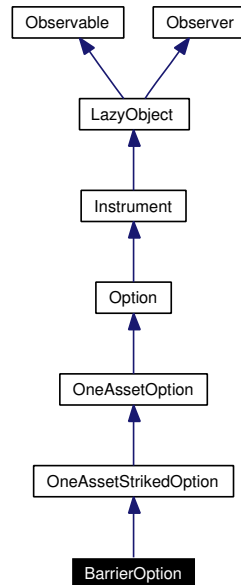
Public Types

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

7.33 BarrierOption Class Reference

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

Inheritance diagram for BarrierOption:



7.33.1 Detailed Description

Barrier option on a single asset.

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

Public Member Functions

- **BarrierOption** (Barrier::Type barrierType, Real barrier, Real rebate, const boost::shared_ptr< BlackScholesProcess > &, const boost::shared_ptr< StrikedTypePayoff > &payoff, const boost::shared_ptr< Exercise > &exercise, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void **setupArguments** (Arguments *) const

Protected Member Functions

- void **performCalculations** () const

Protected Attributes

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

7.33.2 Member Function Documentation

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

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

Reimplemented from [OneAssetStrikedOption](#).

7.33.2.2 void performCalculations () const [protected, virtual]

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

Reimplemented from [OneAssetStrikedOption](#).

7.34 BarrierOption::arguments Class Reference

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

7.34.1 Detailed Description

Arguments for barrier option calculation

Public Member Functions

- void **validate** () const

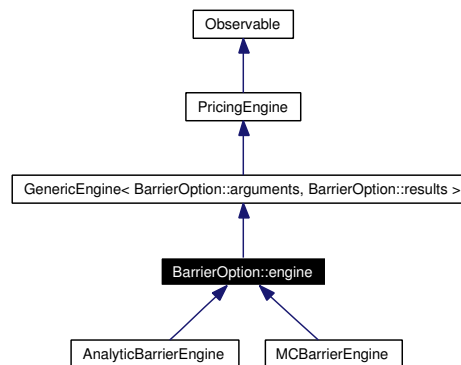
Public Attributes

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

7.35 BarrierOption::engine Class Reference

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

Inheritance diagram for BarrierOption::engine:



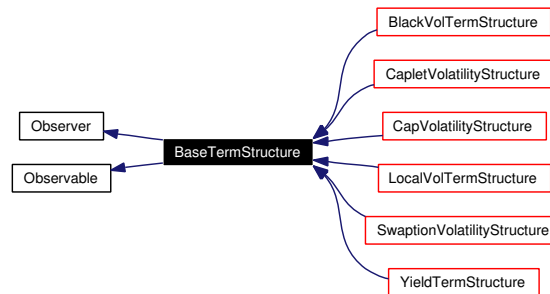
7.35.1 Detailed Description

Barrier engine base class

7.36 BaseTermStructure Class Reference

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

Inheritance diagram for BaseTermStructure:



7.36.1 Detailed Description

Basic term-structure functionality.

Public Member Functions

Constructors

There are three ways in which a term structure can keep track of its reference date. The first is that such date is fixed; the second is that it is determined by advancing the current date of a given number of business days; and the third is that it is based on the reference date of some other structure.

In the first case, the constructor taking a date is to be used; the default implementation of `referenceDate()` will then return such date. In the second case, the constructor taking a number of days and a calendar is to be used; `referenceDate()` will return a date calculated based on the current evaluation date, and the term structure and its observers will be notified when the evaluation date changes. In the last case, the `referenceDate()` method must be overridden in derived classes so that it fetches and return the appropriate date.

- **BaseTermStructure** (const **Date** &todayDate, const **Date** &referenceDate)
initialize with a fixed today's date and reference date
- **BaseTermStructure** ()
default constructor
- **BaseTermStructure** (const **Date** &referenceDate)
initialize with a fixed reference date
- **BaseTermStructure** (**Integer** settlementDays, const **Calendar** &)
calculate the reference date based on the global evaluation date

Dates

- virtual const **Date** & todayDate () const
today's date

- virtual const [Date](#) & [referenceDate](#) () const
the reference date, i.e., the date at which discount = 1
- virtual [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- virtual [DayCounter](#) [dayCounter](#) () const =0
the day counter used for date/time conversion

Observer interface

- void [update](#) ()

Protected Member Functions

- [Time](#) [timeFromReference](#) (const [Date](#) &date) const
date/time conversion

7.36.2 Constructor & Destructor Documentation

7.36.2.1 [BaseTermStructure](#) (const [Date](#) & *today'sDate*, const [Date](#) & *referenceDate*)

initialize with a fixed today's date and reference date

Deprecated

use the constructor without today's date; set the evaluation date through [Settings::instance\(\)](#).

7.36.2.2 [BaseTermStructure](#) ()

default constructor

Warning:

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

7.36.3 Member Function Documentation

7.36.3.1 const [Date](#) & *today'sDate* () const [virtual]

today's date

Deprecated

use [Settings::instance\(\).evaluationDate\(\)](#).

Reimplemented in [DriftTermStructure](#), [ForwardSpreadedTermStructure](#), [QuantoTermStructure](#), and [ZeroSpreadedTermStructure](#).

7.36.3.2 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.

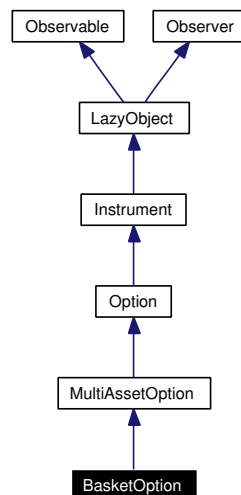
Implements [Observer](#).

Reimplemented in [AffineTermStructure](#), [ExtendedDiscountCurve](#), [PiecewiseFlatForward](#), and [CapVolatilityVector](#).

7.37 BasketOption Class Reference

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

Inheritance diagram for BasketOption:



7.37.1 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< boost::shared_ptr< [BlackScholesProcess](#) > > &stochProcs, const boost::shared_ptr< [PlainVanillaPayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const [Matrix](#) &correlation, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) *) const

7.37.2 Member Function Documentation

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

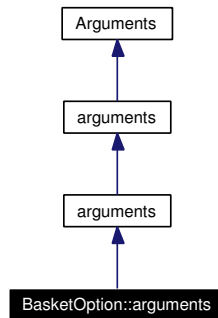
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [MultiAssetOption](#).

7.38 BasketOption::arguments Class Reference

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

Inheritance diagram for BasketOption::arguments:



7.38.1 Detailed Description

Arguments for basket option calculation

Public Member Functions

- void **validate** () const

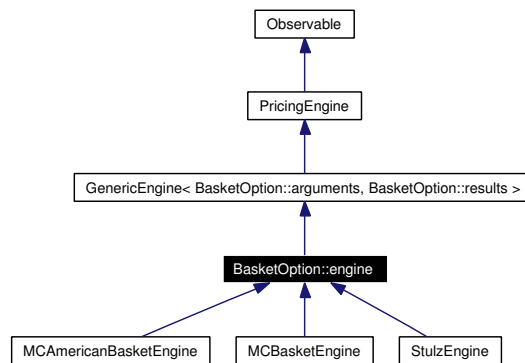
Public Attributes

- BasketType **basketType**

7.39 BasketOption::engine Class Reference

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

Inheritance diagram for BasketOption::engine:



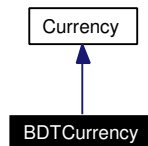
7.39.1 Detailed Description

Basket option engine base class

7.40 BDTCurrency Class Reference

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

Inheritance diagram for BDTCurrency:



7.40.1 Detailed Description

Bangladesh taka.

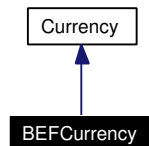
The ISO three-letter code is BDT; the numeric code is 50. It is divided in 100 paisa.

ingroup currencies

7.41 BEFCurrency Class Reference

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

Inheritance diagram for BEFCurrency:



7.41.1 Detailed Description

Belgian franc.

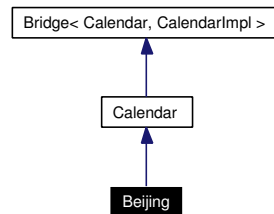
The ISO three-letter code is BEF; the numeric code is 56. It has no subdivisions.

ingroup currencies

7.42 Beijing Class Reference

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

Inheritance diagram for Beijing:



7.42.1 Detailed Description

Beijing calendar

Holidays:

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

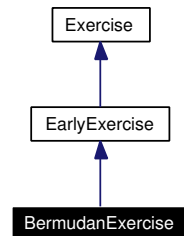
Other holidays for which no rule is given:

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

7.43 BermudanExercise Class Reference

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

Inheritance diagram for BermudanExercise:



7.43.1 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

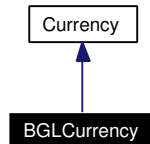
Public Member Functions

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

7.44 BGLCurrency Class Reference

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

Inheritance diagram for BGLCurrency:



7.44.1 Detailed Description

Bulgarian lev.

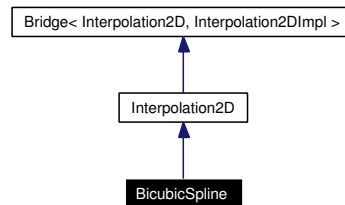
The ISO three-letter code is BGL; the numeric code is 100. It is divided in 100 stotinki.

ingroup currencies

7.45 BicubicSpline Class Reference

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

Inheritance diagram for BicubicSpline:



7.45.1 Detailed Description

bicubic spline interpolation between discrete points

Todo

revise end conditions

Public Member Functions

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

7.45.2 Constructor & Destructor Documentation

7.45.2.1 [BicubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, const I2 & *yEnd*, const M & *zData*)

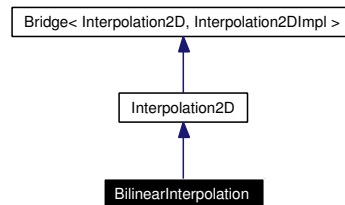
Precondition:

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

7.46 BilinearInterpolation Class Reference

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

Inheritance diagram for BilinearInterpolation:



7.46.1 Detailed Description

bilinear interpolation between discrete points

Public Member Functions

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

7.46.2 Constructor & Destructor Documentation

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

Precondition:

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

7.47 BinomialDistribution Class Reference

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

7.47.1 Detailed Description

Binomial probability distribution function.

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

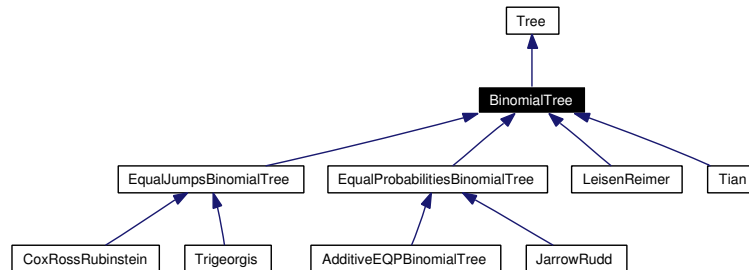
Public Member Functions

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

7.48 BinomialTree Class Reference

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

Inheritance diagram for BinomialTree:



7.48.1 Detailed Description

Binomial tree base class.

Public Member Functions

- **BinomialTree** (const boost::shared_ptr< [StochasticProcess](#) > &process, [Time](#) end, [Size](#) steps)
- [Size](#) size ([Size](#) i) const
- [Size](#) descendant ([Size](#), [Size](#) index, [Size](#) branch) const
- virtual [Real](#) underlying ([Size](#) i, [Size](#) index) const =0
- virtual [Real](#) probability ([Size](#) i, [Size](#) index, [Size](#) branch) const =0

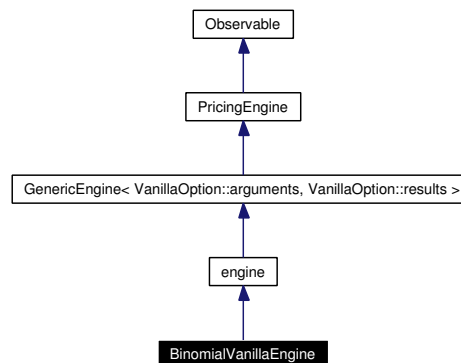
Protected Attributes

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

7.49 BinomialVanillaEngine Class Template Reference

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

Inheritance diagram for BinomialVanillaEngine:



7.49.1 Detailed Description

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

Pricing engine for vanilla options using binomial trees.

Tests

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

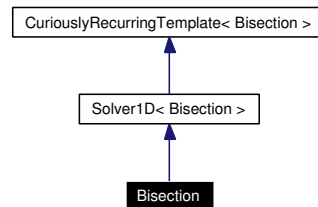
Public Member Functions

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

7.50 Bisection Class Reference

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

Inheritance diagram for Bisection:



7.50.1 Detailed Description

Bisection 1-D solver

Tests

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

Public Member Functions

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

7.51 BivariateCumulativeNormalDistribution Class Reference

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

7.51.1 Detailed Description

Cumulative bivariate normal distribution function.

Drezner (1978) algorithm, six decimal places accuracy.

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

Todo

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

Tests

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

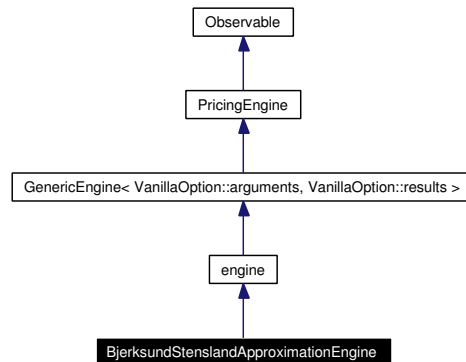
Public Member Functions

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

7.52 BjersundStenslandApproximationEngine Class Reference

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

Inheritance diagram for BjersundStenslandApproximationEngine:



7.52.1 Detailed Description

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

Tests

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

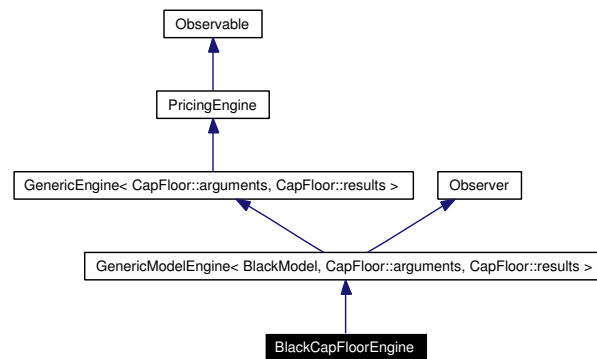
Public Member Functions

- `void calculate () const`

7.53 BlackCapFloorEngine Class Reference

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

Inheritance diagram for BlackCapFloorEngine:



7.53.1 Detailed Description

Black-formula cap/floor engine.

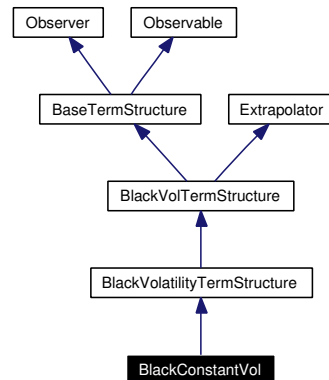
Public Member Functions

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

7.54 BlackConstantVol Class Reference

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

Inheritance diagram for BlackConstantVol:



7.54.1 Detailed Description

Constant Black volatility, no time-strike dependence.

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

Public Member Functions

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

BlackVolTermStructure interface

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

Visitability

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

Protected Member Functions

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

7.55 BlackFormula Class Reference

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

7.55.1 Detailed Description

Black-formula calculator.

Bug

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

Public Member Functions

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

Sensitivity in percent to a percent movement in the underlying.

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

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

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

7.55.2 Member Function Documentation

7.55.2.1 [Real](#) itmCashProbability () const

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

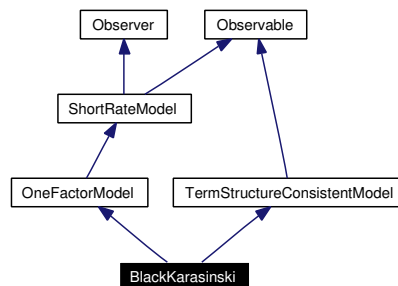
7.55.2.2 [Real](#) itmAssetProbability () const

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

7.56 BlackKarasinski Class Reference

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

Inheritance diagram for BlackKarasinski:



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

7.57 BlackKarasinski::Dynamics Class Reference

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

7.57.1 Detailed Description

Short-rate dynamics in the Black-Karasinski model.

The short-rate is here

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

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

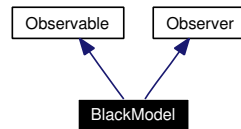
Public Member Functions

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

7.58 BlackModel Class Reference

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

Inheritance diagram for BlackModel:



7.58.1 Detailed Description

Black-model for vanilla interest-rate derivatives.

Public Member Functions

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

Static Public Member Functions

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

7.58.2 Member Function Documentation

7.58.2.1 void update () [virtual]

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

Implements [Observer](#).

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

General Black formula.

Returns

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

where

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

and

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

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

In-the-money cash probability.

Returns

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

where

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

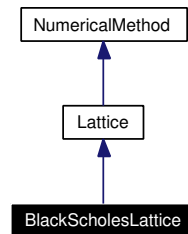
and

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

7.59 BlackScholesLattice Class Reference

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

Inheritance diagram for BlackScholesLattice:



7.59.1 Detailed Description

Simple binomial lattice approximating the Black-Scholes model.

Public Member Functions

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

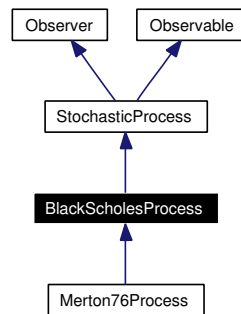
Protected Member Functions

- void **stepback** ([Size](#) i, const [Array](#) &values, [Array](#) &newValues) const
- [Size](#) **descendant** ([Size](#) i, [Size](#) index, [Size](#) branch) const
Tree properties.
- [Real](#) **probability** ([Size](#) i, [Size](#) index, [Size](#) branch) const

7.60 BlackScholesProcess Class Reference

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

Inheritance diagram for BlackScholesProcess:



7.60.1 Detailed Description

Black-Scholes stochastic process.

This class describes the stochastic process governed by

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

Public Member Functions

- **BlackScholesProcess** (const [Handle< Quote >](#) &x0, const [Handle< YieldTermStructure >](#) ÷ndTS, const [Handle< YieldTermStructure >](#) &riskFreeTS, const [Handle< BlackVolTermStructure >](#) &blackVolTS, const boost::shared_ptr< [StochasticProcess::discretization](#) > &d=boost::shared_ptr< [StochasticProcess::discretization](#) >(new [EulerDiscretization](#)))

StochasticProcess interface

- [Real](#) x0 () const
returns the initial value of the state variable
- [Real](#) drift ([Time](#) t, [Real](#) x) const
- [Real](#) diffusion ([Time](#) t, [Real](#) x) const
- [Real](#) evolve ([Real](#) change, [Real](#) currentValue) const

Observer interface

- void [update](#) ()

Inspectors

- const boost::shared_ptr< [Quote](#) > &stateVariable () const
- const boost::shared_ptr< [YieldTermStructure](#) > ÷ndYield () const
- const boost::shared_ptr< [YieldTermStructure](#) > &riskFreeRate () const
- const boost::shared_ptr< [BlackVolTermStructure](#) > &blackVolatility () const
- const boost::shared_ptr< [LocalVolTermStructure](#) > &localVolatility () const

7.60.2 Member Function Documentation

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

[Todo](#)

revise extrapolation

Implements [StochasticProcess](#).

Reimplemented in [Merton76Process](#).

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

[Todo](#)

revise extrapolation

Implements [StochasticProcess](#).

Reimplemented in [Merton76Process](#).

7.60.2.3 [Real](#) evolve ([Real](#) $change$, [Real](#) $currentValue$) const [virtual]

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

Reimplemented from [StochasticProcess](#).

Reimplemented in [Merton76Process](#).

7.60.2.4 void update () [virtual]

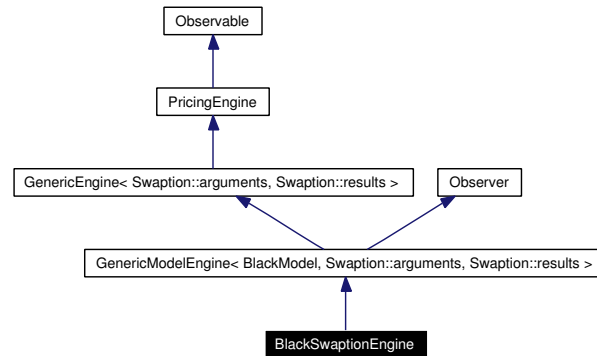
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [StochasticProcess](#).

7.61 BlackSwaptionEngine Class Reference

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

Inheritance diagram for BlackSwaptionEngine:



7.61.1 Detailed Description

Black-formula swaption engine.

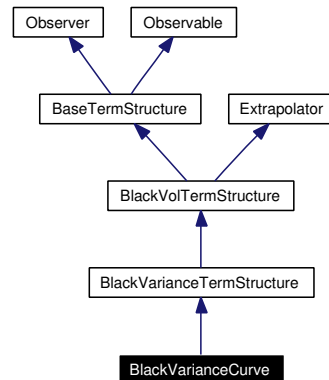
Public Member Functions

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

7.62 BlackVarianceCurve Class Reference

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

Inheritance diagram for BlackVarianceCurve:



7.62.1 Detailed Description

Black volatility curve modelled as variance curve.

This class calculates time-dependent Black volatilities using as input a vector of (ATM) Black volatilities observed in the market.

The calculation is performed interpolating on the variance curve. [Linear](#) interpolation is used as default; this can be changed by the `setInterpolation()` method.

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

Todo

check time extrapolation

Public Member Functions

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

BlackVolTermStructure interface

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

Modifiers

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

Visitability

- `virtual void accept (AcyclicVisitor &)`

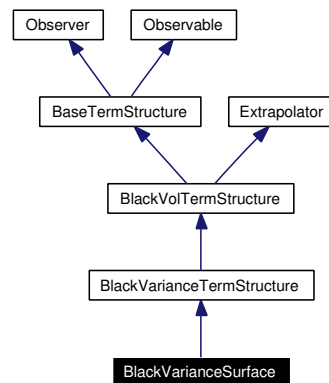
Protected Member Functions

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

7.63 BlackVarianceSurface Class Reference

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

Inheritance diagram for BlackVarianceSurface:



7.63.1 Detailed Description

Black volatility surface modelled as variance surface.

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

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

Todo

check time extrapolation

Public Types

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

Public Member Functions

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

BlackVolTermStructure interface

- `DayCounter dayCounter () const`
the day counter used for date/time conversion

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

Modifiers

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

Visitability

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

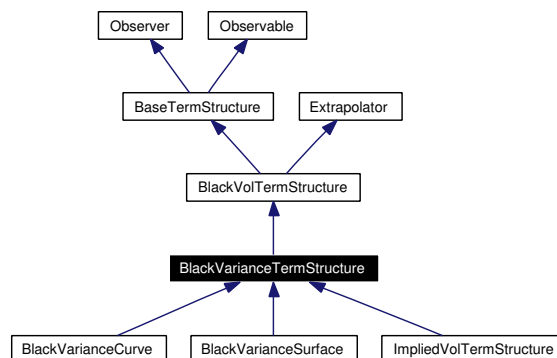
Protected Member Functions

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

7.64 BlackVarianceTermStructure Class Reference

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

Inheritance diagram for BlackVarianceTermStructure:



7.64.1 Detailed Description

Black variance term structure.

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

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

See the *BaseTermStructure* documentation for issues regarding constructors.

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

Visitability

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

Protected Member Functions

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

7.64.2 Constructor & Destructor Documentation

7.64.2.1 [BlackVarianceTermStructure](#) ()

default constructor

Warning:

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

7.64.3 Member Function Documentation

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

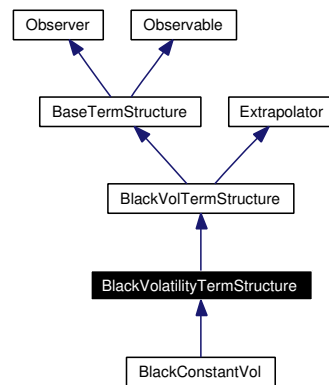
Returns the volatility for the given strike and date calculating it from the variance.

Implements [BlackVolTermStructure](#).

7.65 BlackVolatilityTermStructure Class Reference

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

Inheritance diagram for BlackVolatilityTermStructure:



7.65.1 Detailed Description

Black-volatility term structure.

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

Volatility are assumed to be expressed on an annual basis.

Public Member Functions

Constructors

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

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

Visitability

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

Protected Member Functions

- [Real](#) **blackVarianceImpl** ([Time](#) maturity, [Real](#) strike) const

7.65.2 Constructor & Destructor Documentation

7.65.2.1 [BlackVolatilityTermStructure](#) ()

default constructor

Warning:

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

7.65.3 Member Function Documentation

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

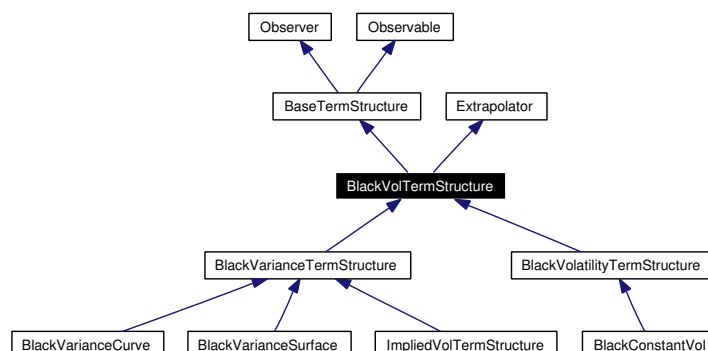
Returns the variance for the given strike and date calculating it from the volatility.

Implements [BlackVolTermStructure](#).

7.66 BlackVolTermStructure Class Reference

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

Inheritance diagram for BlackVolTermStructure:



7.66.1 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

- virtual [DayCounter](#) [dayCounter](#) () const =0
the day counter used for date/time conversion

Constructors

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

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

Black Volatility

- [Volatility](#) [blackVol](#) (const [Date](#) &maturity, [Real](#) strike, bool extrapolate=false) const

present (a.k.a spot) volatility

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

Limits

- virtual **Date maxDate** () const =0
the latest date for which the term structure can return vols
- **Time maxTime** () const
the latest time for which the term structure can return vols
- virtual **Real minStrike** () const =0
the minimum strike for which the term structure can return vols
- virtual **Real maxStrike** () const =0
the maximum strike for which the term structure can return vols

Visitability

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

Protected Member Functions

Calculations

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

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

7.66.2 Constructor & Destructor Documentation

7.66.2.1 [BlackVolTermStructure](#) ()

default constructor

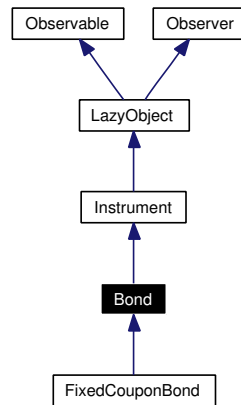
Warning:

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

7.67 Bond Class Reference

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

Inheritance diagram for Bond:



7.67.1 Detailed Description

Base bond class.

Derived classes must fill the uninitialized data members.

Warning:

Most methods assume that the cashflows are stored sorted by date

Tests

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

Instrument interface

- bool [isExpired](#) () const
returns whether the instrument is still tradable.
- void [performCalculations](#) () const

Public Member Functions

- [Date](#) [settlementDate](#) () const
- const std::vector< boost::shared_ptr< [CashFlow](#) > > & [cashflows](#) () const
- const [Calendar](#) & [calendar](#) () const
- const [DayCounter](#) & [dayCounter](#) () const
- [Real](#) [cleanPrice](#) ([Rate](#) yield, [Date](#) settlementDate=[Date](#)()) const
clean price given a yield and settlement date
- [Real](#) [dirtyPrice](#) ([Rate](#) yield, [Date](#) settlementDate=[Date](#)()) const

dirty price given a yield and settlement date

- **Real yield** (**Real** cleanPrice, **Date** settlementDate=**Date**(), **Real** accuracy=1.0e-8, **Size** maxEvaluations=100) const

yield given a (clean) price and settlement date

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

accrued amount at a given date

Protected Member Functions

- **Bond** (const **DayCounter** &dayCount, const **Calendar** &calendar, **Integer** settlementDays)

Protected Attributes

- **Integer** settlementDays_
- **Calendar** calendar_
- **DayCounter** dayCount_
- **Date** issueDate_
- **Date** datedDate_
- **Date** maturityDate_
- **Real** redemption_
- std::vector< boost::shared_ptr< **CashFlow** > > cashFlows_

7.67.2 Member Function Documentation

7.67.2.1 **Real** cleanPrice (**Rate** yield, **Date** settlementDate = **Date**()) const

clean price given a yield and settlement date

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

7.67.2.2 **Real** dirtyPrice (**Rate** yield, **Date** settlementDate = **Date**()) const

dirty price given a yield and settlement date

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

7.67.2.3 **Real** yield (**Real** cleanPrice, **Date** settlementDate = **Date**(), **Real** accuracy = 1.0e-8, **Size** maxEvaluations = 100) const

yield given a (clean) price and settlement date

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

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

accrued amount at a given date

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

7.67.2.5 `void performCalculations () const` [protected, virtual]

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

Reimplemented from [Instrument](#).

7.68 BoundaryCondition Class Template Reference

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

7.68.1 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

7.68.2 Member Enumeration Documentation

7.68.2.1 enum [Side](#)

[Todo](#)

Generalize for n-dimensional conditions

7.68.3 Member Function Documentation

7.68.3.1 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.

7.68.3.2 virtual void [applyAfterApplying](#) (arrayType &) const [pure virtual]

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

7.68.3.3 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.

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

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

7.68.3.5 virtual void setTime ([Time](#) t) [pure virtual]

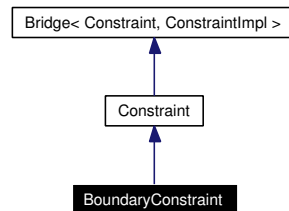
This method sets the current time for time-dependent boundary conditions.

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

7.69 BoundaryConstraint Class Reference

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

Inheritance diagram for BoundaryConstraint:



7.69.1 Detailed Description

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

Public Member Functions

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

7.70 BoxMullerGaussianRng Class Template Reference

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

7.70.1 Detailed Description

```
template<class RNG> class QuantLib::BoxMullerGaussianRng< RNG >
```

Gaussian random number generator.

It uses the well-known Box-Muller transformation to return a normal distributed Gaussian deviate with average 0.0 and standard deviation of 1.0, from a uniform deviate in (0,1) supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Public Types

- typedef [Sample< Real >](#) `sample_type`
- typedef RNG `urng_type`

Public Member Functions

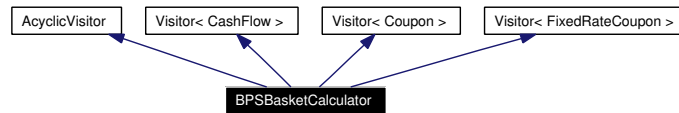
- `BoxMullerGaussianRng` (const RNG &uniformGenerator)
- [sample_type next](#) () const

returns a sample from a Gaussian distribution

7.71 BPSBasketCalculator Class Reference

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

Inheritance diagram for BPSBasketCalculator:



7.71.1 Detailed Description

Bug

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

Public Member Functions

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

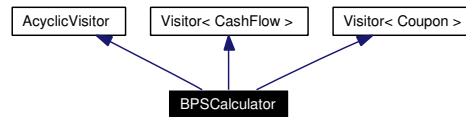
Visitor interface

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

7.72 BPSCalculator Class Reference

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

Inheritance diagram for BPSCalculator:



7.72.1 Detailed Description

basis point sensitivity (BPS) calculator

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

Public Member Functions

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

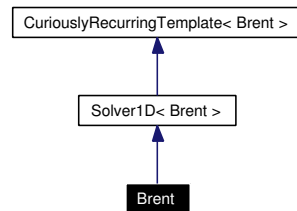
Visitor interface

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

7.73 Brent Class Reference

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

Inheritance diagram for Brent:



7.73.1 Detailed Description

Brent 1-D solver

Tests

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

Public Member Functions

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

7.74 Bridge Class Template Reference

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

7.74.1 Detailed Description

```
template<class T, class T_impl> class QuantLib::Bridge< T, T_impl >
```

The Bridge pattern made explicit.

The typical use of this class is:

```
class FooImpl;
class Foo : public Bridge<Foo,FooImpl> {
    ...
};
```

which makes it possible to pass instances of class Foo by value while retaining polymorphic behavior.

Public Types

- typedef T_impl Impl

Public Member Functions

- bool isNull () const

Protected Member Functions

- Bridge (const boost::shared_ptr< Impl > &impl=boost::shared_ptr< Impl >())

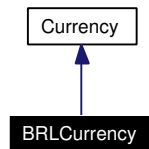
Protected Attributes

- boost::shared_ptr< Impl > impl_

7.75 BRLCurrency Class Reference

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

Inheritance diagram for BRLCurrency:



7.75.1 Detailed Description

Brazilian real.

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

ingroup currencies

7.76 BrownianBridge Class Template Reference

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

7.76.1 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< [Real](#) > > **sample_type**

Public Member Functions

- [BrownianBridge](#) (const GSG &generator)
normalised (unit time, unit variance) Wiener process paths
- [BrownianBridge](#) ([Time](#) length, [Size](#) timeSteps, const GSG &generator)
unit variance Wiener process paths
- [BrownianBridge](#) (const [TimeGrid](#) &timeGrid, const GSG &generator)
unit variance Wiener process paths
- [BrownianBridge](#) (const std::vector< [Real](#) > &sigma, const [TimeGrid](#) &timeGrid, const GSG &generator)
general Wiener process paths
- [BrownianBridge](#) (const boost::shared_ptr< [BlackVolTermStructure](#) > &, const [TimeGrid](#) &timeGrid, const GSG &generator)
- [BrownianBridge](#) (const boost::shared_ptr< [StochasticProcess](#) > &, const [TimeGrid](#) &timeGrid, const GSG &generator)

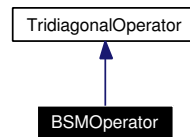
inspectors

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

7.77 BSMOperator Class Reference

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

Inheritance diagram for BSMOperator:



7.77.1 Detailed Description

Black-Scholes-Merton differential operator.

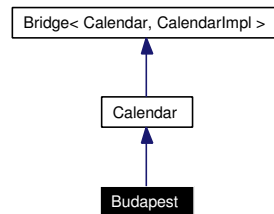
Public Member Functions

- `BSMOperator` ([Size](#) size, [Real](#) dx, [Rate](#) r, [Rate](#) q, [Volatility](#) sigma)

7.78 Budapest Class Reference

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

Inheritance diagram for Budapest:



7.78.1 Detailed Description

Budapest calendar

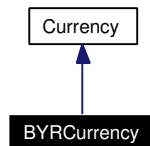
Holidays:

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

7.79 BYRCurrency Class Reference

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

Inheritance diagram for BYRCurrency:



7.79.1 Detailed Description

Belarussian ruble.

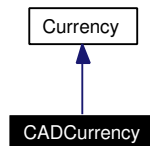
The ISO three-letter code is BYR; the numeric code is 974. It has no subdivisions.

ingroup currencies

7.80 CADCurrency Class Reference

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

Inheritance diagram for CADCurrency:



7.80.1 Detailed Description

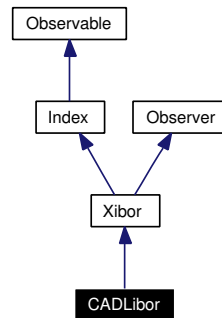
Canadian dollar.

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

7.81 CADLibor Class Reference

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

Inheritance diagram for CADLibor:



7.81.1 Detailed Description

CAD Libor index, also known as CDOR

Todo

check settlement days

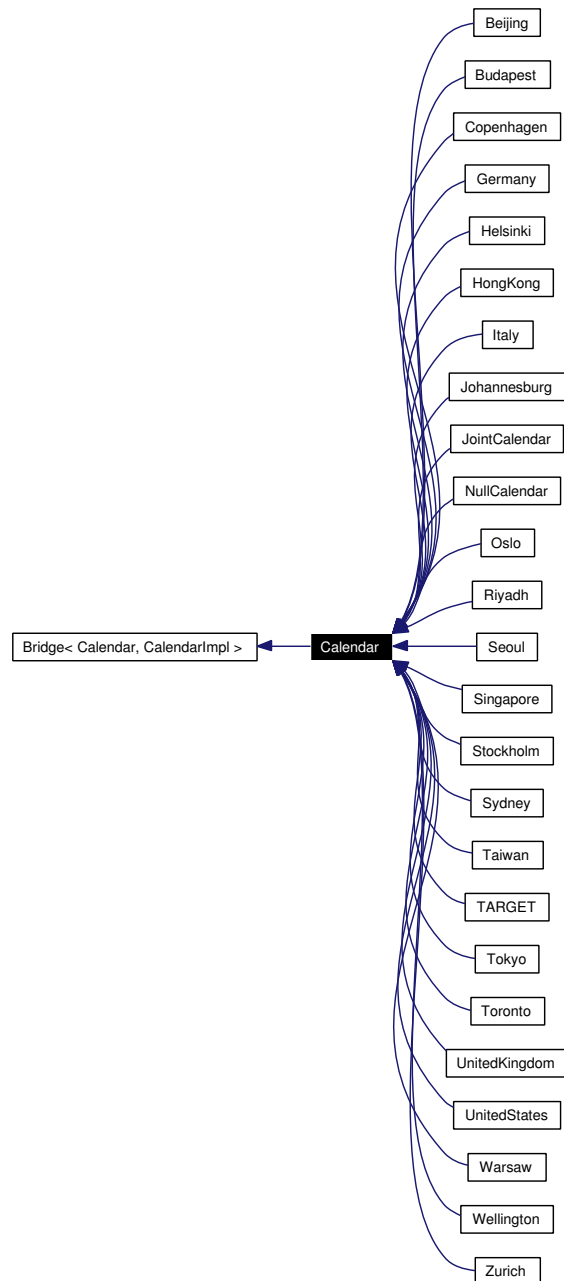
Public Member Functions

- CADLibor (Integer n, TimeUnit units, const Handle< YieldTermStructure > &h, const Day-Counter &dc=Actual365Fixed())

7.82 Calendar Class Reference

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

Inheritance diagram for Calendar:



7.82.1 Detailed Description

calendar class

This class provides methods for determining whether a date is a business day or a holiday for a

given market, and for incrementing/decrementing a date of a given number of business days.

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

A calendar should be defined for specific exchange holiday schedule or for general country holiday schedule. Legacy city holiday schedule calendars will be moved to the exchange/country convention.

Tests

the methods for adding and removing holidays are tested by inspecting the calendar before and after their invocation.

Public Member Functions

- [Calendar](#) ()

Calendar interface

- `std::string name () const`
Returns the name of the calendar.
- `bool isBusinessDay (const Date &d) const`
- `bool isHoliday (const Date &d) const`
- `bool isEndOfMonth (const Date &d) const`
- `void addHoliday (const Date &)`
- `void removeHoliday (const Date &)`
- `Date adjust (const Date &, BusinessDayConvention convention=Following, const Date &origin=Date()) const`
- `Date advance (const Date &, Integer n, TimeUnit unit, BusinessDayConvention convention=Following) const`
- `Date advance (const Date &date, const Period &period, BusinessDayConvention convention=Following) const`

Protected Member Functions

- `Calendar (const boost::shared_ptr< CalendarImpl > &impl)`

Related Functions

(Note that these are not member functions.)

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

7.82.2 Constructor & Destructor Documentation

7.82.2.1 [Calendar](#) ()

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

7.82.2.2 `Calendar` (`const boost::shared_ptr< CalendarImpl > & impl`) [protected]

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

7.82.3 Member Function Documentation

7.82.3.1 `std::string name () const`

Returns the name of the calendar.

Warning:

This method is used for output and comparison between calendars. It is **not** meant to be used for writing switch-on-type code.

7.82.3.2 `bool isBusinessDay (const Date & d) const`

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

7.82.3.3 `bool isHoliday (const Date & d) const`

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

7.82.3.4 `bool isEndOfMonth (const Date & d) const`

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

7.82.3.5 `void addHoliday (const Date &)`

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

7.82.3.6 `void removeHoliday (const Date &)`

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

7.82.3.7 `Date adjust (const Date &, BusinessDayConvention convention = Following, const Date & origin = Date()) const`

Adjusts a non-business day to the appropriate near business day with respect to the given convention.

7.82.3.8 `Date advance (const Date &, Integer n, TimeUnit unit, BusinessDayConvention convention = Following) const`

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

Note:

The input date is not modified.

7.82.3.9 `Date` advance (const `Date` & *date*, const `Period` & *period*, `BusinessDayConvention` *convention* = Following) const

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

Note:

The input date is not modified.

7.82.4 Friends And Related Function Documentation

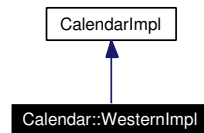
7.82.4.1 `bool operator==(const Calendar &, const Calendar &)` [related]

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

7.83 Calendar::WesternImpl Class Reference

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

Inheritance diagram for Calendar::WesternImpl:



7.83.1 Detailed Description

partial calendar implementation

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

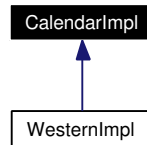
Static Protected Member Functions

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

7.84 CalendarImpl Class Reference

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

Inheritance diagram for CalendarImpl:



7.84.1 Detailed Description

abstract base class for calendar implementations

Public Member Functions

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

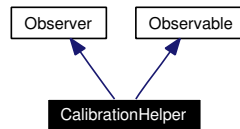
Public Attributes

- `std::set< Date > addedHolidays`
- `std::set< Date > removedHolidays`

7.85 CalibrationHelper Class Reference

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

Inheritance diagram for CalibrationHelper:



7.85.1 Detailed Description

liquid market instrument used during calibration

Public Member Functions

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

Protected Attributes

- [Real](#) [marketValue_](#)
- [Handle](#)< [Quote](#) > [volatility_](#)
- [Handle](#)< [YieldTermStructure](#) > [termStructure_](#)
- boost::shared_ptr< [BlackModel](#) > [blackModel_](#)
- boost::shared_ptr< [PricingEngine](#) > [engine_](#)

7.85.2 Member Function Documentation

7.85.2.1 void update () [virtual]

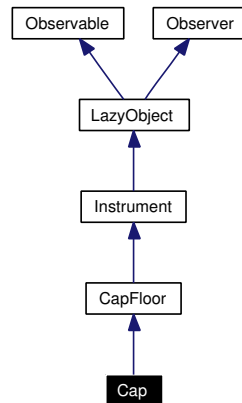
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.86 Cap Class Reference

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

Inheritance diagram for Cap:



7.86.1 Detailed Description

Concrete cap class.

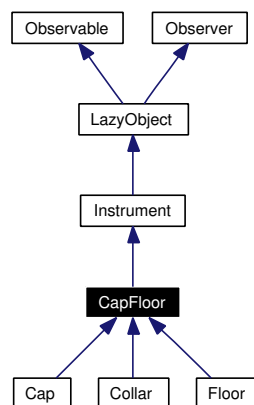
Public Member Functions

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

7.87 CapFloor Class Reference

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

Inheritance diagram for CapFloor:



7.87.1 Detailed Description

Base class for cap-like instruments.

Tests

- the correctness of the returned value is tested by checking that the price of a cap (resp. floor) decreases (resp. increases) with the strike rate.
- the relationship between the values of caps, floors and the resulting collars is checked.
- the put-call parity between the values of caps, floors and swaps is checked.
- the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- the correctness of the returned value is tested by checking it against a known good value.

Public Types

- enum Type { Cap, Floor, Collar }

Public Member Functions

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

Instrument interface

- bool [isExpired](#) () const
returns whether the instrument is still tradable.

Inspectors

- Type **type** () const
- const std::vector< boost::shared_ptr< [CashFlow](#) > > & **leg** () const
- const std::vector< [Rate](#) > & **capRates** () const
- const std::vector< [Rate](#) > & **floorRates** () const

7.87.2 Member Function Documentation

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

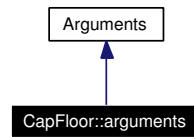
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

7.88 CapFloor::arguments Class Reference

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

Inheritance diagram for CapFloor::arguments:



7.88.1 Detailed Description

Arguments for cap/floor calculation

Public Member Functions

- void **validate** () const

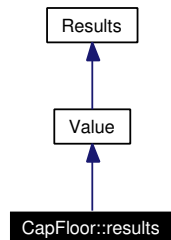
Public Attributes

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

7.89 CapFloor::results Class Reference

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

Inheritance diagram for CapFloor::results:



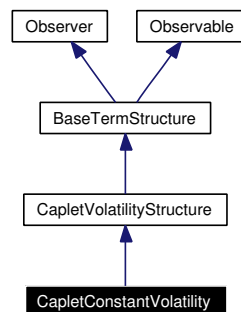
7.89.1 Detailed Description

Results from cap/floor calculation

7.90 CapletConstantVolatility Class Reference

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

Inheritance diagram for CapletConstantVolatility:



7.90.1 Detailed Description

Constant caplet volatility, no time-strike dependence.

Public Member Functions

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

BaseTermStructure interface

- [DayCounter](#) dayCounter () const
the day counter used for date/time conversion

Protected Member Functions

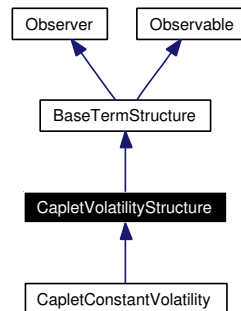
CapletVolatilityStructure interface

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

7.91 CapletVolatilityStructure Class Reference

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

Inheritance diagram for CapletVolatilityStructure:



7.91.1 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 [DayCounter](#) [dayCounter](#) () const =0
the day counter used for date/time conversion

Constructors

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

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

Volatility

- [Volatility](#) [volatility](#) (const [Date](#) &start, [Rate](#) strike) const
returns the volatility for a given start date and strike rate
- [Volatility](#) [volatility](#) ([Time](#) t, [Rate](#) strike) const
returns the volatility for a given start time and strike rate

Protected Member Functions

- virtual [Volatility volatilityImpl](#) ([Time](#) length, [Rate](#) strike) const =0
implements the actual volatility calculation in derived classes

7.91.2 Constructor & Destructor Documentation

7.91.2.1 [CapletVolatilityStructure](#) ()

default constructor

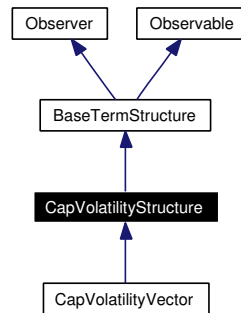
Warning:

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

7.92 CapVolatilityStructure Class Reference

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

Inheritance diagram for CapVolatilityStructure:



7.92.1 Detailed Description

Cap/floor term-volatility structure.

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

Public Member Functions

- virtual [DayCounter](#) [dayCounter](#) () const =0
the day counter used for date/time conversion

Constructors

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

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

Volatility

- [Volatility](#) [volatility](#) (const [Date](#) &end, [Rate](#) strike) const
- [Volatility](#) [volatility](#) (const [Period](#) &length, [Rate](#) strike) const
returns the volatility for a given cap/floor length and strike rate

- [Volatility volatility](#) ([Time](#) t, [Rate](#) strike) const
returns the volatility for a given end time and strike rate

Protected Member Functions

- virtual [Volatility volatilityImpl](#) ([Time](#) length, [Rate](#) strike) const =0
implements the actual volatility calculation in derived classes

7.92.2 Constructor & Destructor Documentation

7.92.2.1 [CapVolatilityStructure](#) ()

default constructor

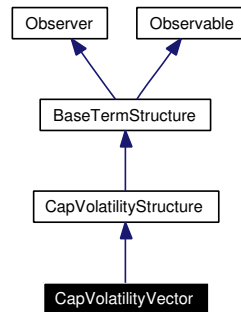
Warning:

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

7.93 CapVolatilityVector Class Reference

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

Inheritance diagram for CapVolatilityVector:



7.93.1 Detailed Description

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

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

Todo

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

Public Member Functions

- **CapVolatilityVector** (const [Date](#) &todayDate, const [Calendar](#) &calendar, [Integer](#) settlementDays, const std::vector< [Period](#) > &lengths, const std::vector< [Volatility](#) > &volatilities, const [DayCounter](#) &dayCounter)
- **CapVolatilityVector** (const [Date](#) &settlementDate, const std::vector< [Period](#) > &lengths, const std::vector< [Volatility](#) > &volatilities, const [DayCounter](#) &dayCounter)
- **CapVolatilityVector** ([Integer](#) settlementDays, const [Calendar](#) &calendar, const std::vector< [Period](#) > &lengths, const std::vector< [Volatility](#) > &volatilities, const [DayCounter](#) &dayCounter)
- [DayCounter](#) dayCounter () const
the day counter used for date/time conversion
- void [update](#) ()

7.93.2 Constructor & Destructor Documentation

- 7.93.2.1 **CapVolatilityVector** (const [Date](#) & todayDate, const [Calendar](#) & calendar, [Integer](#) settlementDays, const std::vector< [Period](#) > & lengths, const std::vector< [Volatility](#) > & volatilities, const [DayCounter](#) & dayCounter)

Deprecated

use one of the other constructors

7.93.3 Member Function Documentation

7.93.3.1 void update () [virtual]

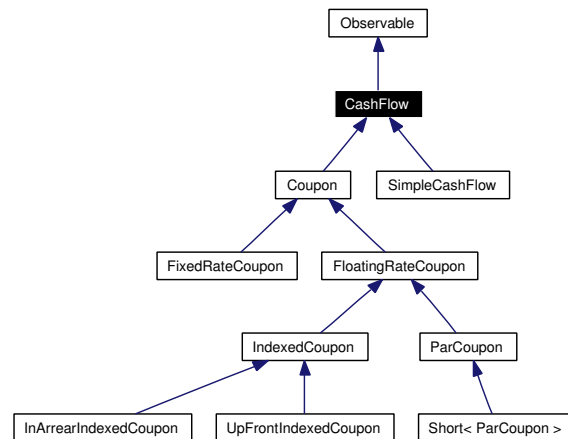
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [BaseTermStructure](#).

7.94 CashFlow Class Reference

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

Inheritance diagram for CashFlow:



7.94.1 Detailed Description

Base class for cash flows.

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

Public Member Functions

CashFlow interface

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

Visitability

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

7.94.2 Member Function Documentation

7.94.2.1 virtual **Real amount** () const [pure virtual]

returns the amount of the cash flow

Note:

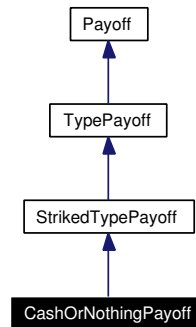
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implemented in [FixedRateCoupon](#), [IndexedCoupon](#), [ParCoupon](#), [Short< ParCoupon >](#), and [SimpleCashFlow](#).

7.95 CashOrNothingPayoff Class Reference

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

Inheritance diagram for CashOrNothingPayoff:



7.95.1 Detailed Description

Binary cash-or-nothing payoff.

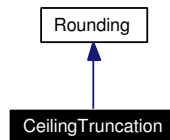
Public Member Functions

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

7.96 CeilingTruncation Class Reference

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

Inheritance diagram for CeilingTruncation:



7.96.1 Detailed Description

Ceiling truncation.

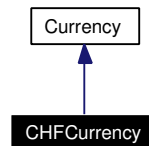
Public Member Functions

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

7.97 CHFCurrency Class Reference

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

Inheritance diagram for CHFCurrency:



7.97.1 Detailed Description

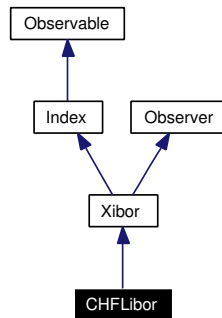
Swiss franc.

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

7.98 CHFLibor Class Reference

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

Inheritance diagram for CHFLibor:



7.98.1 Detailed Description

CHF Libor index, also known as ZIBOR

Todo

check settlement days and day-count

Public Member Functions

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

7.99 CLGaussianRng Class Template Reference

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

7.99.1 Detailed Description

```
template<class RNG> class QuantLib::CLGaussianRng< RNG >
```

Gaussian random number generator.

It uses the well-known fact that the sum of 12 uniform deviate in $(-.5,.5)$ is approximately a Gaussian deviate with average 0 and standard deviation 1. The uniform deviate is supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Public Types

- typedef [Sample](#)< [Real](#) > **sample_type**
- typedef RNG **urng_type**

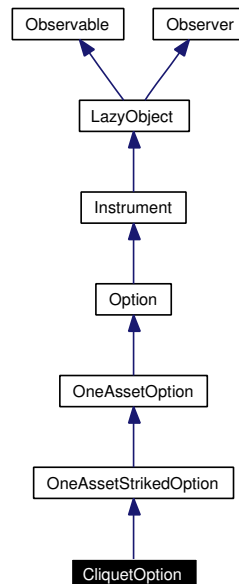
Public Member Functions

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

7.100 CliquetOption Class Reference

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

Inheritance diagram for CliquetOption:



7.100.1 Detailed Description

cliquet (Ratchet) option

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

Todo

- add local/global caps/floors
- add accrued coupon and last fixing

Public Member Functions

- **CliquetOption** (const boost::shared_ptr< [BlackScholesProcess](#) > &, const boost::shared_ptr< [PercentageStrikePayoff](#) > &, const boost::shared_ptr< [EuropeanExercise](#) > & maturity, const std::vector< [Date](#) > &resetDates, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) *) const

7.100.2 Member Function Documentation

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

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

Reimplemented from [OneAssetStrikedOption](#).

7.101 CliquetOption::arguments Class Reference

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

7.101.1 Detailed Description

Arguments for cliquet option calculation

Public Member Functions

- void **validate** () const

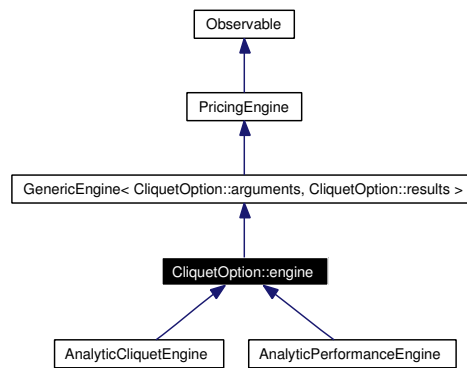
Public Attributes

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

7.102 CliquetOption::engine Class Reference

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

Inheritance diagram for CliquetOption::engine:



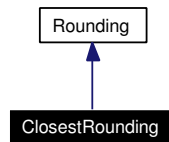
7.102.1 Detailed Description

Cliquet engine base class.

7.103 ClosestRounding Class Reference

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

Inheritance diagram for ClosestRounding:



7.103.1 Detailed Description

Closest rounding.

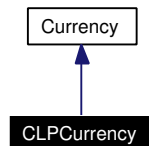
Public Member Functions

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

7.104 CLPCurrency Class Reference

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

Inheritance diagram for CLPCurrency:



7.104.1 Detailed Description

Chilean peso.

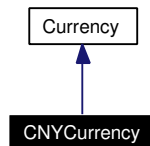
The ISO three-letter code is CLP; the numeric code is 152. It is divided in 100 centavos.

ingroup currencies

7.105 CNYCurrency Class Reference

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

Inheritance diagram for CNYCurrency:



7.105.1 Detailed Description

Chinese yuan.

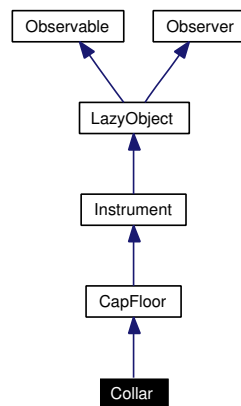
The ISO three-letter code is CNY; the numeric code is 156. It is divided in 100 fen.

ingroup currencies

7.106 Collar Class Reference

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

Inheritance diagram for Collar:



7.106.1 Detailed Description

Concrete collar class.

Public Member Functions

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

7.107 combining_iterator Class Template Reference

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

7.107.1 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/>)

Deprecated

use a combination of `boost::zip_iterator` and `boost::transform_iterator` instead

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

7.108 Composite Class Template Reference

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

7.108.1 Detailed Description

template<class T> class QuantLib::Composite< T >

Composite pattern.

The typical use of this class is:

```
class CompositeFoo : public Composite<Foo> {  
    ...  
};
```

which causes CompositeFoo to inherit from Foo and provides it with methods for adding components. Of course, any abstract Foo interface must still be implemented.

Protected Types

- typedef std::list< boost::shared_ptr< T > >::iterator **iterator**
- typedef std::list< boost::shared_ptr< T > >::const_iterator **const_iterator**

Protected Member Functions

- void **add** (const boost::shared_ptr< T > &c)

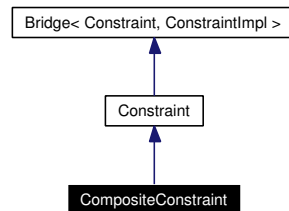
Protected Attributes

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

7.109 CompositeConstraint Class Reference

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

Inheritance diagram for CompositeConstraint:



7.109.1 Detailed Description

Constraint enforcing both given sub-constraints

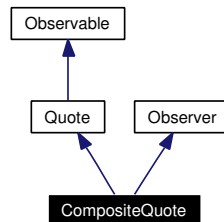
Public Member Functions

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

7.110 CompositeQuote Class Template Reference

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

Inheritance diagram for CompositeQuote:



7.110.1 Detailed Description

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

market element whose value depends on two other market element

Tests

the correctness of the returned values is tested by checking them against numerical calculations.

Public Member Functions

- **CompositeQuote** (const [Handle](#)< [Quote](#) > &element1, const [Handle](#)< [Quote](#) > &element2, const BinaryFunction &f)

Quote interface

- [Real value](#) () const
returns the current value

Observer interface

- void [update](#) ()

7.110.2 Member Function Documentation

7.110.2.1 void update () [virtual]

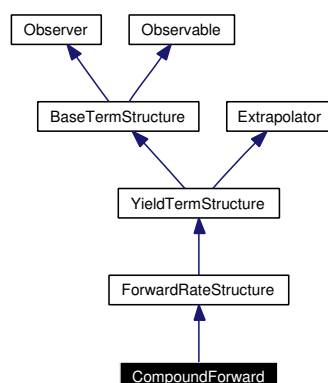
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.111 CompoundForward Class Reference

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

Inheritance diagram for CompoundForward:



7.111.1 Detailed Description

compound-forward structure

Tests

- the correctness of the curve is tested by reproducing the input data.
- the correctness of the curve is tested by checking the consistency between returned rates and swaps priced on the curve.

Bug

swap rates are not reproduced exactly when using indexed coupons. Apparently, some assumption about the swap fixings is hard-coded into the bootstrapping algorithm. the class does not operate correctly when `QL_DISABLE_DEPRECATED` is defined. Investigation is required.

Public Member Functions

- **CompoundForward** (const [Date](#) &todayDate, const [Date](#) &referenceDate, const std::vector< [Date](#) > &dates, const std::vector< [Rate](#) > &forwards, const [Calendar](#) &calendar, const [BusinessDayConvention](#) conv, const [Integer](#) compounding, const [DayCounter](#) &dayCounter)
- **CompoundForward** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &dates, const std::vector< [Rate](#) > &forwards, const [Calendar](#) &calendar, const [BusinessDayConvention](#) conv, const [Integer](#) compounding, const [DayCounter](#) &dayCounter)
- **Calendar** calendar () const
the calendar used for reference date calculation
- **BusinessDayConvention** businessDayConvention () const
- **DayCounter** dayCounter () const
the day counter used for date/time conversion

- [Integer](#) **compounding** () const
- [Date](#) **maxDate** () const
the latest date for which the curve can return rates
- [Time](#) **maxTime** () const
the latest time for which the curve can return rates
- const std::vector< [Time](#) > & **times** () const
- const std::vector< [Date](#) > & **dates** () const
- const std::vector< [Rate](#) > & **forwards** () const
- boost::shared_ptr< [YieldTermStructure](#) > **discountCurve** () const

Protected Member Functions

- void **calibrateNodes** () const
- boost::shared_ptr< [YieldTermStructure](#) > **bootstrap** () const
- [Rate](#) **zeroYieldImpl** ([Time](#)) const
- [DiscountFactor](#) **discountImpl** ([Time](#)) const
- [Size](#) **referenceNode** ([Time](#)) const
- [Rate](#) **forwardImpl** ([Time](#)) const
instantaneous forward-rate calculation
- [Rate](#) **compoundForwardImpl** ([Time](#), [Integer](#)) const

7.111.2 Constructor & Destructor Documentation

- 7.111.2.1 [CompoundForward](#) (const [Date](#) & *today'sDate*, const [Date](#) & *referenceDate*, const std::vector< [Date](#) > & *dates*, const std::vector< [Rate](#) > & *forwards*, const [Calendar](#) & *calendar*, const [BusinessDayConvention](#) *conv*, const [Integer](#) *compounding*, const [DayCounter](#) & *dayCounter*)

Deprecated

use the constructor without today's date

7.111.3 Member Function Documentation

- 7.111.3.1 [Rate](#) **zeroYieldImpl** ([Time](#)) const [protected, virtual]

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

Warning:

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

Reimplemented from [ForwardRateStructure](#).

- 7.111.3.2 [DiscountFactor](#) **discountImpl** ([Time](#)) const [protected, virtual]

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

Reimplemented from [ForwardRateStructure](#).

7.111.3.3 [Rate](#) compoundForwardImpl ([Time](#), [Integer](#)) const [protected, virtual]

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

Reimplemented from [ForwardRateStructure](#).

7.112 CompoundingRuleFormatter Class Reference

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

7.112.1 Detailed Description

Formats compounding rule for output.

Uses [FrequencyFormatter](#) and adds compounding informations

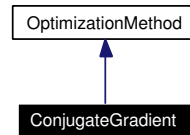
Static Public Member Functions

- `std::string toString` (Compounding comp, [Frequency](#) freq=NoFrequency)

7.113 ConjugateGradient Class Reference

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

Inheritance diagram for ConjugateGradient:



7.113.1 Detailed Description

Multi-dimensional Conjugate Gradient class.

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

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

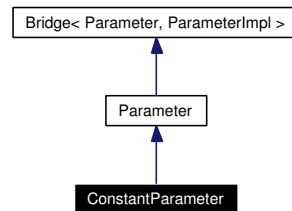
Public Member Functions

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

7.114 ConstantParameter Class Reference

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

Inheritance diagram for ConstantParameter:



7.114.1 Detailed Description

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

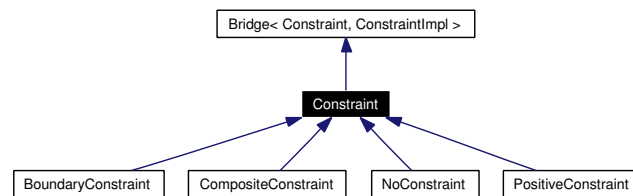
Public Member Functions

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

7.115 Constraint Class Reference

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

Inheritance diagram for Constraint:



7.115.1 Detailed Description

Base constraint class.

Public Member Functions

- **bool test** (const [Array](#) &p) const
- **Real update** ([Array](#) &p, const [Array](#) &direction, [Real](#) beta)
- **Constraint** (const boost::shared_ptr< [ConstraintImpl](#) > &impl=boost::shared_ptr< [ConstraintImpl](#) >())

7.116 ConstraintImpl Class Reference

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

7.116.1 Detailed Description

Base class for constraint implementations.

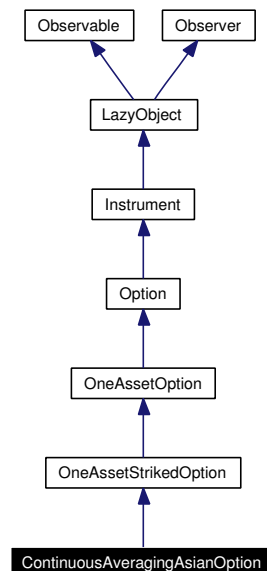
Public Member Functions

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

7.117 ContinuousAveragingAsianOption Class Reference

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

Inheritance diagram for ContinuousAveragingAsianOption:



7.117.1 Detailed Description

Continuous-averaging Asian option.

Todo

add running average

Public Member Functions

- **ContinuousAveragingAsianOption** (Average::Type averageType, const boost::shared_ptr< [BlackScholesProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) *) const

Protected Attributes

- Average::Type [averageType_](#)

7.117.2 Member Function Documentation

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

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

Reimplemented from [OneAssetStrikedOption](#).

7.118 ContinuousAveragingAsianOption::arguments Class Reference

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

7.118.1 Detailed Description

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

Public Member Functions

- void **validate** () const

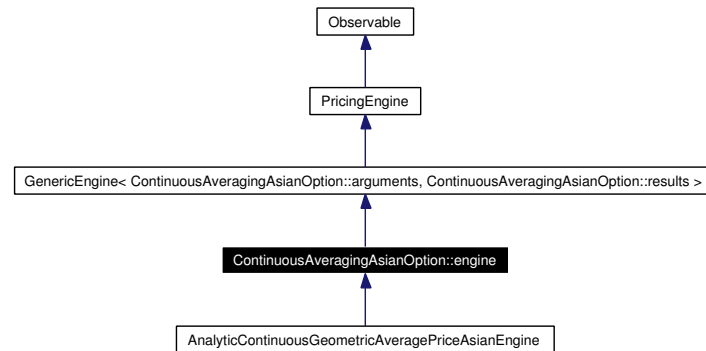
Public Attributes

- Average::Type **averageType**

7.119 ContinuousAveragingAsianOption::engine Class Reference

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

Inheritance diagram for ContinuousAveragingAsianOption::engine:



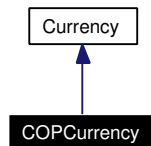
7.119.1 Detailed Description

Continuous-averaging Asian engine base class.

7.120 COPCurrency Class Reference

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

Inheritance diagram for COPCurrency:



7.120.1 Detailed Description

Colombian peso.

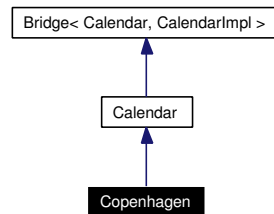
The ISO three-letter code is COP; the numeric code is 170. It is divided in 100 centavos.

ingroup currencies

7.121 Copenhagen Class Reference

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

Inheritance diagram for Copenhagen:



7.121.1 Detailed Description

Copenhagen calendar

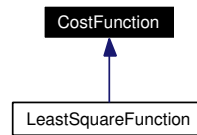
Holidays:

- Saturdays
- Sundays
- Maunday Thursday
- Good Friday
- Easter Monday
- General Prayer Day, 25 days after Easter Monday
- Ascension
- Whit (Pentecost) Monday
- New Year's Day, January 1st
- Constitution Day, June 5th
- Christmas, December 25th
- Boxing Day, December 26th

7.122 CostFunction Class Reference

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

Inheritance diagram for CostFunction:



7.122.1 Detailed Description

Cost function abstract class for optimization problem.

Public Member Functions

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

7.123 coupling_iterator Class Template Reference

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

7.123.1 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.

Deprecated

use a combination of `boost::zip_iterator` and `boost::transform_iterator` instead

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

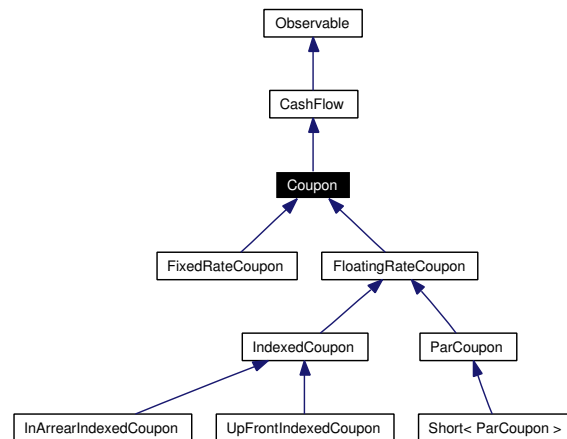
(Note that these are not member functions.)

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

7.124 Coupon Class Reference

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

Inheritance diagram for Coupon:



7.124.1 Detailed Description

coupon accruing over a fixed period

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

Public Member Functions

- [Coupon](#) ([Real](#) nominal, const [Date](#) &paymentDate, const [Date](#) &accrualStartDate, const [Date](#) &accrualEndDate, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)())

Partial CashFlow interface

- [Date](#) [date](#) () const
returns the date at which the cash flow is settled

Inspectors

- [Real](#) [nominal](#) () const
- const [Date](#) & [accrualStartDate](#) () const
start of the accrual period
- const [Date](#) & [accrualEndDate](#) () const
end of the accrual period
- [Time](#) [accrualPeriod](#) () const
accrual period as fraction of year
- [Integer](#) [accrualDays](#) () const

accrual period in days

- virtual `Rate rate () const =0`
accrued rate
- virtual `DayCounter dayCounter () const =0`
day counter for accrual calculation
- virtual `Real accruedAmount (const Date &) const =0`
accrued amount at the given date

Visitability

- virtual void `accept (AcyclicVisitor &)`

Protected Attributes

- `Real nominal_`
- `Date paymentDate_`
- `Date accrualStartDate_`
- `Date accrualEndDate_`
- `Date refPeriodStart_`
- `Date refPeriodEnd_`

7.124.2 Constructor & Destructor Documentation

7.124.2.1 **Coupon** (`Real nominal`, `const Date & paymentDate`, `const Date & accrualStartDate`, `const Date & accrualEndDate`, `const Date & refPeriodStart = Date()`, `const Date & refPeriodEnd = Date()`)

Warning:

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

7.125 CovarianceDecomposition Class Reference

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

7.125.1 Detailed Description

Extracts the correlation matrix and the vector of volatilities out of the input covariance matrix. Note that only the lower symmetric part of the covariance matrix is used.

Precondition:

The covariance matrix must be symmetric.

Tests

cross checked with getCovariance

Public Member Functions

- [CovarianceDecomposition](#) (const [Matrix](#) &covarianceMatrix, [Real](#) tolerance=1.0e-12)
- const [Array](#) & [variances](#) () const
- const [Array](#) & [standardDeviations](#) () const
- const [Matrix](#) & [correlationMatrix](#) () const

7.125.2 Constructor & Destructor Documentation

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

Precondition:

covarianceMatrix must be symmetric

7.125.3 Member Function Documentation

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

returns the variances [Array](#)

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

returns the standard deviations [Array](#)

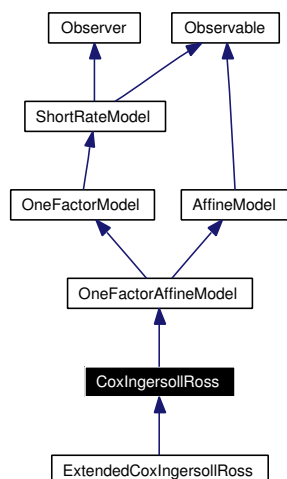
7.125.3.3 const [Matrix](#)& [correlationMatrix](#) () const

returns the correlation matrix

7.126 CoxIngersollRoss Class Reference

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

Inheritance diagram for CoxIngersollRoss:



7.126.1 Detailed Description

Cox-Ingersoll-Ross model class.

This class implements the Cox-Ingersoll-Ross model defined by

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

Bug

this class was not tested enough to guarantee its functionality.

Public Member Functions

- **CoxIngersollRoss** (**Rate** r0=0.05, **Real** theta=0.1, **Real** k=0.1, **Real** sigma=0.1)
- virtual **Real** **discountBondOption** (Option::Type type, **Real** strike, **Time** maturity, **Time** bondMaturity) const
- virtual boost::shared_ptr< ShortRateDynamics > **dynamics** () const
returns the short-rate dynamics
- virtual boost::shared_ptr< **Lattice** > **tree** (const **TimeGrid** &grid) const
Return by default a trinomial recombining tree.

Protected Member Functions

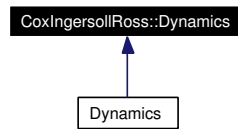
- **Real** **A** (**Time** t, **Time** T) const
- **Real** **B** (**Time** t, **Time** T) const
- **Real** **theta** () const

- [Real k](#) () const
- [Real sigma](#) () const
- [Real x0](#) () const

7.127 CoxIngersollRoss::Dynamics Class Reference

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

Inheritance diagram for CoxIngersollRoss::Dynamics:



7.127.1 Detailed Description

Dynamics of the short-rate under the Cox-Ingersoll-Ross model

The state variable y_t will here be the square-root of the short-rate. It satisfies the following stochastic equation

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

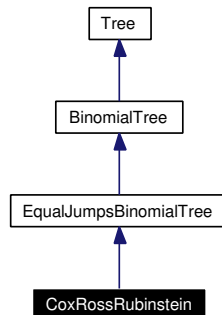
Public Member Functions

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

7.128 CoxRossRubinstein Class Reference

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

Inheritance diagram for CoxRossRubinstein:



7.128.1 Detailed Description

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

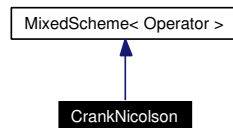
Public Member Functions

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

7.129 CrankNicolson Class Template Reference

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

Inheritance diagram for CrankNicolson:



7.129.1 Detailed Description

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

Crank-Nicolson scheme for finite difference methods.

See sect. [Finite-differences framework](#) for details on the method.

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

```

typedef ... 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*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);

```

Warning:

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

Friends

- class `FiniteDifferenceModel<CrankNicolson<Operator> >`

7.130 Cubic Class Reference

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

7.130.1 Detailed Description

Cubic-spline interpolation traits.

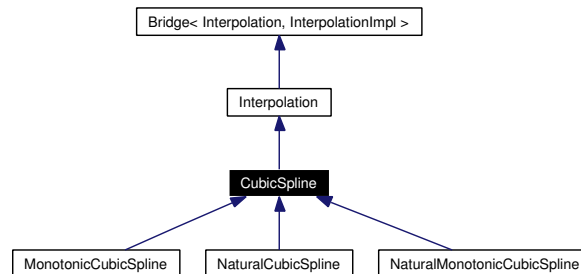
Static Public Member Functions

- `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)

7.131 CubicSpline Class Reference

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

Inheritance diagram for CubicSpline:



7.131.1 Detailed Description

Cubic spline interpolation between discrete points.

It implements different type of end conditions: not-a-knot, first derivative value, second derivative value.

It also implements Hyman's monotonicity constraint filter which ensures that the interpolating spline remains monotonic at the expense of the second derivative of the curve which will no longer be continuous where the filter has been applied. If the interpolating spline is already monotonic, the Hyman filter leaves it unchanged.

See R. L. Dougherty, A. Edelman, and J. M. Hyman, "Nonnegativity-, Monotonicity-, or Convexity-Preserving Cubic and Quintic Hermite Interpolation" Mathematics Of Computation, v. 52, n. 186, April 1989, pp. 471-494.

Tests

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

Public Types

- enum [BoundaryCondition](#) {
[NotAKnot](#), [FirstDerivative](#), [SecondDerivative](#), [Periodic](#),
[Lagrange](#) }

Public Member Functions

- template<class I1, class I2> [CubicSpline](#) (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, [CubicSpline::BoundaryCondition](#) leftCondition, [Real](#) leftConditionValue, [CubicSpline::BoundaryCondition](#) rightCondition, [Real](#) rightConditionValue, bool monotonicity-Constraint)
- const std::vector< [Real](#) > & [aCoefficients](#) () const
- const std::vector< [Real](#) > & [bCoefficients](#) () const
- const std::vector< [Real](#) > & [cCoefficients](#) () const

7.131.2 Member Enumeration Documentation

7.131.2.1 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

7.131.3 Constructor & Destructor Documentation

7.131.3.1 [CubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, [CubicSpline::BoundaryCondition](#) *leftCondition*, [Real](#) *leftConditionValue*, [CubicSpline::BoundaryCondition](#) *rightCondition*, [Real](#) *rightConditionValue*, bool *monotonicityConstraint*)

Precondition:

the *x* values must be sorted.

7.132 CumulativeBinomialDistribution Class Reference

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

7.132.1 Detailed Description

Cumulative binomial distribution function.

Given an integer k it provides the cumulative probability of observing $k \leq k$: formula here ...

Public Member Functions

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

7.133 CumulativeNormalDistribution Class Reference

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

7.133.1 Detailed Description

Cumulative normal distribution function.

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

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

Public Member Functions

- **CumulativeNormalDistribution** ([Real](#) average=0.0, [Real](#) sigma=1.0)
- [Real](#) **operator()** ([Real](#) x) const
- [Real](#) **derivative** ([Real](#) x) const

7.134 CumulativePoissonDistribution Class Reference

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

7.134.1 Detailed Description

Cumulative Poisson distribution function.

This function provides an approximation of the integral of the Poisson distribution.

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

Tests

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

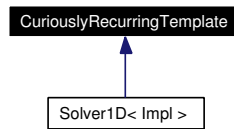
Public Member Functions

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

7.135 CuriouslyRecurringTemplate Class Template Reference

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

Inheritance diagram for CuriouslyRecurringTemplate:



7.135.1 Detailed Description

```
template<class Impl> class QuantLib::CuriouslyRecurringTemplate< Impl >
```

Support for the curiously recurring template pattern.

See James O. Coplien. A Curiously Recurring Template Pattern. In Stanley B. Lippman, editor, C++ Gems, 135-144. Cambridge University Press, New York, New York, 1996.

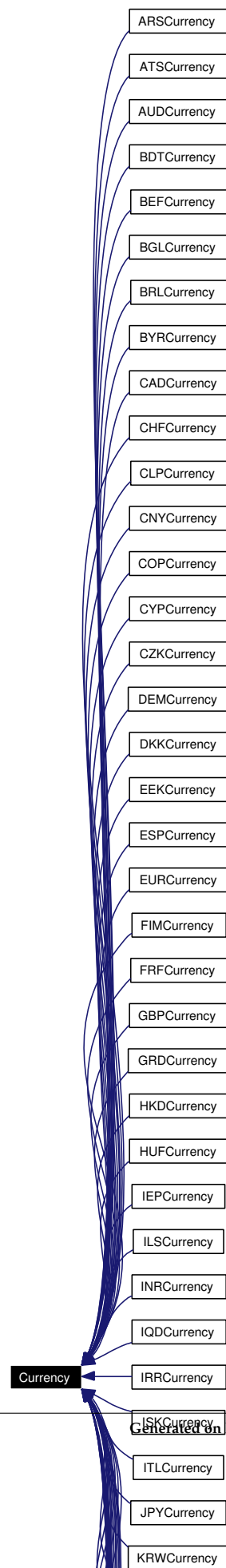
Protected Member Functions

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

7.136 Currency Class Reference

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

Inheritance diagram for Currency:



7.136.1 Detailed Description

Currency specification

Public Member Functions

- [Currency](#) ()
default constructor

Inspectors

- const std::string & [name](#) () const
currency name, e.g, "U.S. Dollar"
- const std::string & [code](#) () const
ISO 4217 three-letter code, e.g, "USD".
- [Integer numericCode](#) () const
ISO 4217 numeric code, e.g, "840".
- const std::string & [symbol](#) () const
symbol, e.g, "\$"
- const std::string & [fractionSymbol](#) () const
fraction symbol, e.g, "162"
- [Integer fractionsPerUnit](#) () const
number of fractionary parts in a unit, e.g, 100
- const [Rounding](#) & [rounding](#) () const
rounding convention
- boost::format [format](#) () const
output format

other info

- bool [isValid](#) () const
is this a usable instance?
- const [Currency](#) & [triangulationCurrency](#) () const
currency used for triangulated exchange when required

Protected Attributes

- boost::shared_ptr< Data > [data_](#)

Related Functions

(Note that these are not member functions.)

- `bool operator==` (const [Currency](#) &, const [Currency](#) &)
- `bool operator!=` (const [Currency](#) &, const [Currency](#) &)

7.136.2 Constructor & Destructor Documentation

7.136.2.1 [Currency](#) ()

default constructor

Instances built via this constructor have undefined behavior. Such instances can only act as placeholders and must be reassigned to a valid currency before being used.

7.136.3 Member Function Documentation

7.136.3.1 `boost::format format () const`

output format

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

7.137 CurrencyFormatter Class Reference

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

7.137.1 Detailed Description

format currencies for output

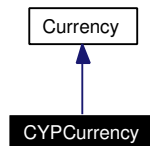
Static Public Member Functions

- `std::string toString` ([CurrencyTag](#) c)
- `std::string toString` (const [Currency](#) &c)

7.138 CYPCurrency Class Reference

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

Inheritance diagram for CYPCurrency:



7.138.1 Detailed Description

Cyprus pound.

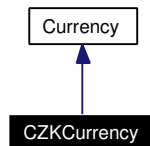
The ISO three-letter code is CYP; the numeric code is 196. It is divided in 100 cents.

ingroup currencies

7.139 CZKCurrency Class Reference

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

Inheritance diagram for CZKCurrency:



7.139.1 Detailed Description

Czech koruna.

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

ingroup currencies

7.140 Date Class Reference

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

7.140.1 Detailed Description

Concrete date class.

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

Tests

self-consistency of dates, serial numbers, days of month, months, and weekdays is checked over the whole date range.

Public Member Functions

constructors

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

inspectors

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

date algebra

- [Date](#) & [operator+=](#) ([BigInteger](#) days)
increments date by the given number of days
- [Date](#) & [operator+=](#) (const [Period](#) &)
increments date by the given period
- [Date](#) & [operator-=](#) ([BigInteger](#) days)
decrement date by the given number of days

- [Date & operator-=](#) (const [Period](#) &)
decrements date by the given period
- [Date & operator++](#) ()
1-day pre-increment
- [Date operator++](#) (int)
1-day post-increment
- [Date & operator--](#) ()
1-day pre-decrement
- [Date operator--](#) (int)
1-day post-decrement
- [Date operator+](#) ([BigInteger](#) days) const
returns a new date incremented by the given number of days
- [Date operator+](#) (const [Period](#) &) const
returns a new date incremented by the given period
- [Date operator-](#) ([BigInteger](#) days) const
returns a new date decremented by the given number of days
- [Date operator-](#) (const [Period](#) &) const
returns a new date decremented by the given period

other methods to increment/decrement dates

- [Date plusDays](#) ([Integer](#) n) const
- [Date plusWeeks](#) ([Integer](#) n) const
- [Date plusMonths](#) ([Integer](#) n) const
- [Date plusYears](#) ([Integer](#) n) const
- [Date plus](#) ([Integer](#) n, [TimeUnit](#) units) const
- [Date plus](#) (const [Period](#) &) const

Static Public Member Functions

static methods

- [Date todaysDate](#) ()
today's date.
- [Date minDate](#) ()
earliest allowed date
- [Date maxDate](#) ()
latest allowed date
- [bool isLeap](#) ([Year](#) y)
whether the given year is a leap one

- [Date endOfMonth](#) (const [Date](#) &d)
last day of the month to which the given date belongs
- bool [isEOM](#) (const [Date](#) &d)
whether a date is the last day of its month
- [Date nextWeekday](#) (const [Date](#) &d, [Weekday](#))
next given weekday following or equal to the given date
- [Date nthWeekday](#) ([Size](#) n, [Weekday](#), [Month](#) m, [Year](#) y)
n-th given weekday in the given month and year
- bool [isIMMdate](#) (const [Date](#) &d)
whether or not the given date is an IMM date
- [Date nextIMMdate](#) (const [Date](#) &d)
next IMM date following (or equal to) the given date

Related Functions

(Note that these are not member functions.)

- `std::ostream & operator<<` (`std::ostream &`, const [Date](#) &)
- [BigInteger operator-](#) (const [Date](#) &, const [Date](#) &)
Difference in days between dates.
- bool `operator==` (const [Date](#) &, const [Date](#) &)
- bool `operator!=` (const [Date](#) &, const [Date](#) &)
- bool `operator<` (const [Date](#) &, const [Date](#) &)
- bool `operator<=` (const [Date](#) &, const [Date](#) &)
- bool `operator>` (const [Date](#) &, const [Date](#) &)
- bool `operator>=` (const [Date](#) &, const [Date](#) &)

7.140.2 Member Function Documentation

7.140.2.1 bool isEndOfMonth () const

Deprecated

use the static [isEOM\(\)](#) method instead

7.140.2.2 [Day](#) lastDayOfMonth () const

Deprecated

use the static [endOfMonth\(\)](#) method instead

7.140.2.3 Date plusDays (Integer *n*) const**Deprecated**

use date + n*Days instead

7.140.2.4 Date plusWeeks (Integer *n*) const**Deprecated**

use date + n*Weeks instead

7.140.2.5 Date plusMonths (Integer *n*) const**Deprecated**

use date + n*Months instead

7.140.2.6 Date plusYears (Integer *n*) const**Deprecated**

use date + n*Years instead

7.140.2.7 Date plus (Integer *n*, TimeUnit *units*) const**Deprecated**

use date + n*units instead

7.140.2.8 Date plus (const Period &) const**Deprecated**

use date + period instead

7.140.2.9 Date nextWeekday (const Date & *d*, Weekday) [static]

next given weekday following or equal to the given date

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

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

7.140.2.10 Date nthWeekday (Size *n*, Weekday, Month *m*, Year *y*) [static]

n-th given weekday in the given month and year

E.g., the 4th Thursday of March, 1998 was March 26th, 1998.

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

7.140.2.11 [Date](#) nextIMMdate (const [Date](#) & *d*) [static]

next IMM date following (or equal to) the given date

returns the 1st delivery date for next contract listed in the International [Money](#) Market section of the Chicago Mercantile Exchange.

Warning:

The result date is following or equal to the original date

Examples:

[swapvaluation.cpp](#).

7.140.3 Friends And Related Function Documentation

7.140.3.1 `std::ostream & operator<< (std::ostream &, const Date &)` [related]

Deprecated

send to the stream the output of [DateFormatter](#)

7.141 DateFormatter Class Reference

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

7.141.1 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 }

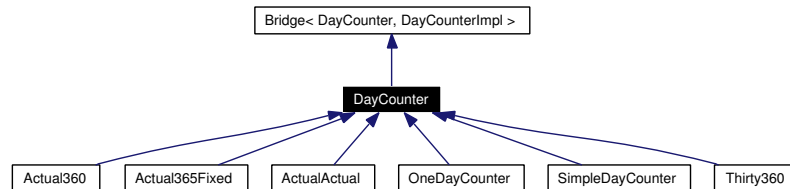
Static Public Member Functions

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

7.142 DayCounter Class Reference

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

Inheritance diagram for DayCounter:



7.142.1 Detailed Description

day counter class

This class provides methods for determining the length of a time period according to given market convention, both as a number of days and as a year fraction.

The [Bridge](#) pattern is used to provide the base behavior of the day counter.

Public Member Functions

- [DayCounter](#) ()

DayCounter interface

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

Protected Member Functions

- `DayCounter (const boost::shared_ptr< DayCounterImpl > &impl)`

Related Functions

(Note that these are not member functions.)

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

7.142.2 Constructor & Destructor Documentation

7.142.2.1 [DayCounter](#) ()

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

7.142.2.2 [DayCounter](#) (const boost::shared_ptr< [DayCounterImpl](#) > & *impl*) [protected]

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

7.142.3 Member Function Documentation

7.142.3.1 std::string name () const

Returns the name of the day counter.

Warning:

This method is used for output and comparison between day counters. It is **not** meant to be used for writing switch-on-type code.

7.142.4 Friends And Related Function Documentation

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

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

7.143 DayCounterImpl Class Reference

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

7.143.1 Detailed Description

abstract base class for day counter implementations

Public Member Functions

- virtual std::string **name** () const =0
- virtual [BigInteger](#) **dayCount** (const [Date](#) &d1, const [Date](#) &d2) const
to be overloaded by more complex day counters
- virtual [Time](#) **yearFraction** (const [Date](#) &, const [Date](#) &, const [Date](#) &refPeriodStart, const [Date](#) &refPeriodEnd) const =0

7.144 DecimalFormatter Class Reference

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

7.144.1 Detailed Description

Formats real numbers for output.

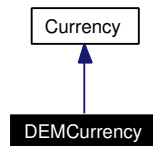
Static Public Member Functions

- `std::string toString` ([Decimal](#) x, [Integer](#) precision=6, [Integer](#) digits=0)
- `std::string toExponential` ([Decimal](#) x, [Integer](#) precision=6, [Integer](#) digits=0)
- `std::string toPercentage` ([Decimal](#) x, [Integer](#) precision=6, [Integer](#) digits=0)

7.145 DEMCurrency Class Reference

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

Inheritance diagram for DEMCurrency:



7.145.1 Detailed Description

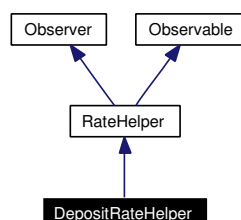
Deutsche mark.

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

7.146 DepositRateHelper Class Reference

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

Inheritance diagram for DepositRateHelper:



7.146.1 Detailed Description

Deposit rate.

Warning:

This class assumes that the reference date does not change between calls of [setTermStructure\(\)](#).

Public Member Functions

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

7.146.2 Member Function Documentation

7.146.2.1 void setTermStructure ([YieldTermStructure](#) *) [virtual]

sets the term structure to be used for pricing

Warning:

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

Reimplemented from [RateHelper](#).

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

latest relevant date

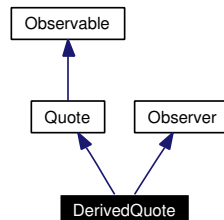
The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

7.147 DerivedQuote Class Template Reference

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

Inheritance diagram for DerivedQuote:



7.147.1 Detailed Description

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

market element whose value depends on another market element

Tests

the correctness of the returned values is tested by checking them against numerical calculations.

Public Member Functions

- **DerivedQuote** (const [Handle](#)< [Quote](#) > &element, const UnaryFunction &f)

Market element interface

- [Real value](#) () const
returns the current value

Observer interface

- void [update](#) ()

7.147.2 Member Function Documentation

7.147.2.1 void update () [virtual]

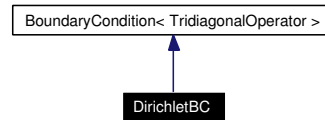
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.148 DirichletBC Class Reference

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Inheritance diagram for DirichletBC:



7.148.1 Detailed Description

Neumann boundary condition (i.e., constant value).

Todo

generalize to time-dependent conditions.

Public Member Functions

- **DirichletBC** ([Real](#) value, Side side)
- void **applyBeforeApplying** ([TridiagonalOperator](#) &) const
- void **applyAfterApplying** ([Array](#) &) const
- void **applyBeforeSolving** ([TridiagonalOperator](#) &, [Array](#) &rhs) const
- void **applyAfterSolving** ([Array](#) &) const
- void **setTime** ([Time](#))

7.148.2 Member Function Documentation

7.148.2.1 void setTime ([Time](#)) [virtual]

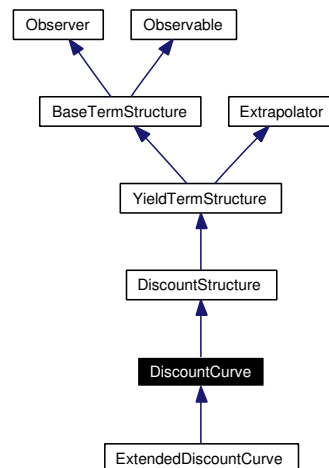
This method sets the current time for time-dependent boundary conditions.

Implements [BoundaryCondition< TridiagonalOperator >](#).

7.149 DiscountCurve Class Reference

```
#include <ql/TermStructures/discountcurve.hpp>
```

Inheritance diagram for DiscountCurve:



7.149.1 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](#) &todayDate, const std::vector< [Date](#) > &dates, const std::vector< [DiscountFactor](#) > &dfs, const [DayCounter](#) &dayCounter)
- **DiscountCurve** (const std::vector< [Date](#) > &dates, const std::vector< [DiscountFactor](#) > &dfs, const [DayCounter](#) &dayCounter)
- **DayCounter** [dayCounter](#) () const
the day counter used for date/time conversion
- **Date** [maxDate](#) () const
the latest date for which the curve can return 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](#)) const
discount calculation
- [Size](#) [referenceNode](#) ([Time](#)) const

Protected Attributes

- [DayCounter](#) [dayCounter_](#)
- `std::vector< Date >` [dates_](#)
- `std::vector< DiscountFactor >` [discounts_](#)
- `std::vector< Time >` [times_](#)
- [Interpolation](#) [interpolation_](#)

7.149.2 Constructor & Destructor Documentation

7.149.2.1 [DiscountCurve](#) (const [Date](#) & *today'sDate*, const `std::vector< Date >` & *dates*, const `std::vector< DiscountFactor >` & *dfs*, const [DayCounter](#) & *dayCounter*)

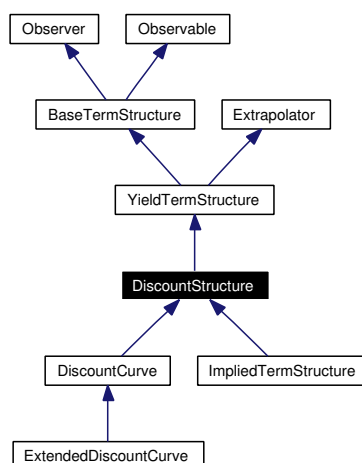
Deprecated

use the constructor without today's date.

7.150 DiscountStructure Class Reference

```
#include <ql/TermStructures/discountstructure.hpp>
```

Inheritance diagram for DiscountStructure:



7.150.1 Detailed Description

Discount factor term structure.

This abstract class acts as an adapter to [YieldTermStructure](#) allowing the programmer to implement only the `discountImpl(const Date&, bool)` method in derived classes. Zero yield and forward are calculated from discounts.

Rates are assumed to be annual continuous compounding.

Deprecated

use [YieldTermStructure](#) instead

Public Member Functions

Constructors

See the [BaseTermStructure](#) documentation for issues regarding constructors.

- **DiscountStructure** (const [Date](#) &todayDate, const [Date](#) &referenceDate)
- **DiscountStructure** ()
- **DiscountStructure** (const [Date](#) &referenceDate)
- **DiscountStructure** ([Integer](#) settlementDays, const [Calendar](#) &)

Protected Member Functions

YieldTermStructure implementation

- [Rate zeroYieldImpl](#) ([Time](#)) const
- [Rate forwardImpl](#) ([Time](#)) const
- virtual [Rate compoundForwardImpl](#) ([Time](#), [Integer](#)) const

7.150.2 Member Function Documentation

7.150.2.1 **Rate** zeroYieldImpl (**Time**) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the discount.

Implements [YieldTermStructure](#).

7.150.2.2 **Rate** forwardImpl (**Time**) const [protected, virtual]

Returns the instantaneous forward rate for the given date calculating it from the discount.

Implements [YieldTermStructure](#).

7.150.2.3 **Rate** compoundForwardImpl (**Time**, **Integer**) const [protected, virtual]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Implements [YieldTermStructure](#).

Reimplemented in [ExtendedDiscountCurve](#).

7.151 DiscrepancyStatistics Class Reference

```
#include <ql/Math/discrepancystatistics.hpp>
```

Inheritance diagram for DiscrepancyStatistics:



7.151.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

It inherit from [SequenceStatistics<Statistics>](#) and adds L^2 discrepancy calculation

Public Member Functions

- **DiscrepancyStatistics** ([Size](#) dimension)
- `template<class Sequence> void add (const Sequence &sample, Real weight=1.0)`
- `template<class Iterator> void add (Iterator begin, Iterator end, Real weight=1.0)`
- `void reset (Size dimension=0)`

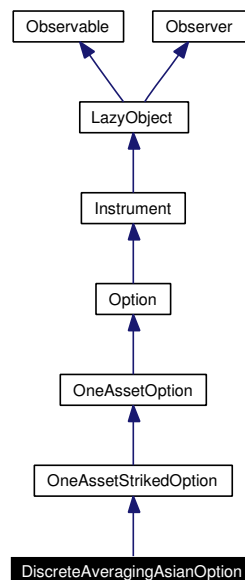
1-dimensional inspectors

- `Real discrepancy () const`

7.152 DiscreteAveragingAsianOption Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

Inheritance diagram for DiscreteAveragingAsianOption:



7.152.1 Detailed Description

Discrete-averaging Asian option.

Public Member Functions

- **DiscreteAveragingAsianOption** (Average::Type averageType, Real runningAccumulator, Size pastFixings, std::vector< Date > fixingDates, const boost::shared_ptr< BlackScholesProcess > &, const boost::shared_ptr< StrikedTypePayoff > &payoff, const boost::shared_ptr< Exercise > &exercise, const boost::shared_ptr< PricingEngine > &engine=boost::shared_ptr< PricingEngine >())
- void **setupArguments** (Arguments *) const

Protected Attributes

- Average::Type **averageType_**
- Real **runningAccumulator_**
- Size **pastFixings_**
- std::vector< Date > **fixingDates_**

7.152.2 Member Function Documentation

7.152.2.1 void setupArguments ([Arguments](#) *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetStrikedOption](#).

7.153 DiscreteAveragingAsianOption::arguments Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

7.153.1 Detailed Description

Extra arguments for single-asset discrete-average Asian option.

Public Member Functions

- void **validate** () const

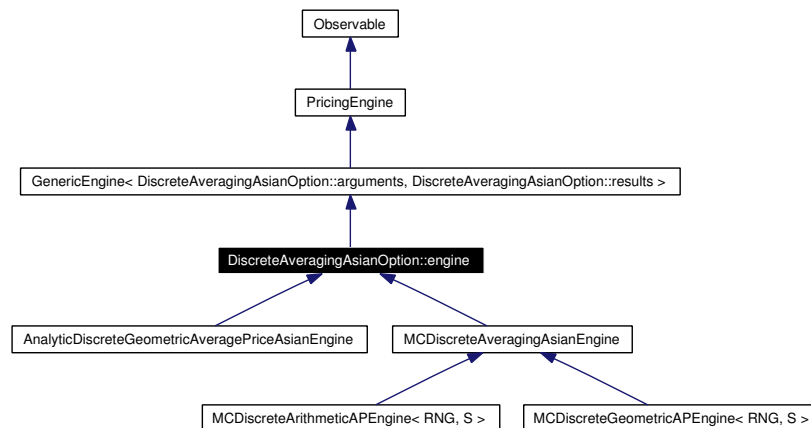
Public Attributes

- Average::Type **averageType**
- [Real](#) **runningAccumulator**
- [Size](#) **pastFixings**
- std::vector< [Date](#) > **fixingDates**

7.154 DiscreteAveragingAsianOption::engine Class Reference

```
#include <ql/Instruments/asianoption.hpp>
```

Inheritance diagram for DiscreteAveragingAsianOption::engine:



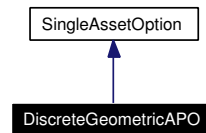
7.154.1 Detailed Description

Discrete-averaging Asian engine base class.

7.155 DiscreteGeometricAPO Class Reference

```
#include <ql/Pricers/discretegeometricapo.hpp>
```

Inheritance diagram for DiscreteGeometricAPO:



7.155.1 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

Deprecated

use the [DiscreteAveragingAsianOption](#) instrument with [AnalyticDiscreteGeometricAveragePriceAsianEngine](#) instead

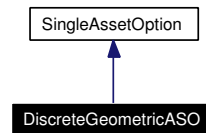
Public Member Functions

- **DiscreteGeometricAPO** (Option::Type type, [Real](#) underlying, [Real](#) strike, [Spread](#) dividend-Yield, [Rate](#) riskFreeRate, const std::vector< [Time](#) > ×, [Volatility](#) volatility)
- [Real](#) **value** () const
- [Real](#) **delta** () const
- [Real](#) **gamma** () const
- [Real](#) **theta** () const
- boost::shared_ptr< [SingleAssetOption](#) > **clone** () const

7.156 DiscreteGeometricASO Class Reference

```
#include <ql/Pricers/discretegeometricaso.hpp>
```

Inheritance diagram for DiscreteGeometricASO:



7.156.1 Detailed Description

Discrete geometric average-strike Asian option (European style).

This class implements a discrete geometric average strike asian option, with european exercise. The formula is from "Asian Option", E. Levy (1997) in "Exotic Options: The State of the Art", edited by L. Clewlow, C. Strickland, pag65-97

Todo

add analytical greeks

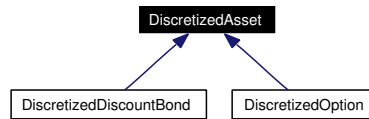
Public Member Functions

- **DiscreteGeometricASO** (Option::Type type, [Real](#) underlying, [Spread](#) dividendYield, [Rate](#) riskFreeRate, const std::vector< [Time](#) > ×, [Volatility](#) volatility)
- [Real](#) **value** () const
- [Real](#) **delta** () const
- [Real](#) **gamma** () const
- [Real](#) **theta** () const
- boost::shared_ptr< [SingleAssetOption](#) > **clone** () const

7.157 DiscretizedAsset Class Reference

```
#include <ql/discretizedasset.hpp>
```

Inheritance diagram for DiscretizedAsset:



7.157.1 Detailed Description

Discretized asset class used by numerical methods.

Public Member Functions

- [DiscretizedAsset](#) (const boost::shared_ptr< [NumericalMethod](#) > &method)

inspectors

- [Time](#) [time](#) () const
- [Time](#) & [time](#) ()
- const [Array](#) & [values](#) () const
- [Array](#) & [values](#) ()
- const boost::shared_ptr< [NumericalMethod](#) > & [method](#) () const

High-level interface

Users of discretized assets should use these methods in order to initialize, evolve and take the present value of the assets.

- void [initialize](#) (const boost::shared_ptr< [NumericalMethod](#) > &, [Time](#) t)
- void [rollback](#) ([Time](#) to)
- void [partialRollback](#) ([Time](#) to)
- [Real](#) [presentValue](#) ()

Low-level interface

These methods (that developers should override when deriving from DiscretizedAsset) are to be used by numerical methods and not directly by users.

- virtual void [reset](#) ([Size](#) size)=0
- void [preAdjustValues](#) ()
- void [postAdjustValues](#) ()
- void [adjustValues](#) ()
- virtual std::vector< [Time](#) > [mandatoryTimes](#) () const
- virtual void [addTimesTo](#) (std::list< [Time](#) > &l) const

Protected Member Functions

- bool [isOnTime](#) ([Time](#) t) const
- virtual void [preAdjustValuesImpl](#) ()
- virtual void [postAdjustValuesImpl](#) ()

Protected Attributes

- [Time](#) `time_`
- [Time](#) `latestPreAdjustment_`
- [Time](#) `latestPostAdjustment_`
- [Array](#) `values_`

7.157.2 Constructor & Destructor Documentation

7.157.2.1 [DiscretizedAsset](#) (`const boost::shared_ptr< NumericalMethod > & method`)

Deprecated

use the constructor with no arguments

7.157.3 Member Function Documentation

7.157.3.1 `virtual void reset (Size size)` [pure virtual]

This method should initialize the asset values to an [Array](#) of the given size and with values depending on the particular asset.

Implemented in [DiscretizedDiscountBond](#), and [DiscretizedOption](#).

7.157.3.2 `void preAdjustValues ()`

This method will be invoked after rollback and before any other asset (i.e., an option on this one) has any chance to look at the values. For instance, payments happening at times already spanned by the rollback will be added here.

This method is not virtual; derived classes must override the protected [preAdjustValuesImpl\(\)](#) method instead.

7.157.3.3 `void postAdjustValues ()`

This method will be invoked after rollback and after any other asset had their chance to look at the values. For instance, payments happening at the present time (and therefore not included in an option to be exercised at this time) will be added here.

This method is not virtual; derived classes must override the protected [postAdjustValuesImpl\(\)](#) method instead.

7.157.3.4 `void adjustValues ()`

This method performs both pre- and post-adjustment

7.157.3.5 `virtual std::vector<Time> mandatoryTimes () const` [virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

Note:

The returned values are not guaranteed to be sorted.

Reimplemented in [DiscretizedOption](#).

7.157.3.6 virtual void addTimesTo (std::list< [Time](#) > & l) const [virtual]

This method appends to the given list the times at which the numerical method should stop while rolling back.

[Deprecated](#)

use [mandatoryTimes\(\)](#) instead.

7.157.3.7 bool isOnTime ([Time](#) t) const [protected]

This method checks whether the asset was rolled at the given time.

7.157.3.8 virtual void preAdjustValuesImpl () [protected, virtual]

This method performs the actual pre-adjustment

7.157.3.9 virtual void postAdjustValuesImpl () [protected, virtual]

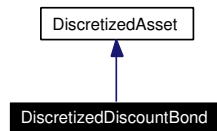
This method performs the actual post-adjustment

Reimplemented in [DiscretizedOption](#).

7.158 DiscretizedDiscountBond Class Reference

```
#include <ql/discretizedasset.hpp>
```

Inheritance diagram for DiscretizedDiscountBond:



7.158.1 Detailed Description

Useful discretized discount bond asset.

Public Member Functions

- [DiscretizedDiscountBond](#) (const boost::shared_ptr< [NumericalMethod](#) > &method)
- void [reset](#) ([Size](#) size)

7.158.2 Constructor & Destructor Documentation

- 7.158.2.1 [DiscretizedDiscountBond](#) (const boost::shared_ptr< [NumericalMethod](#) > &
method)

Deprecated

use the constructor with no arguments

7.158.3 Member Function Documentation

- 7.158.3.1 void [reset](#) ([Size](#) size) [virtual]

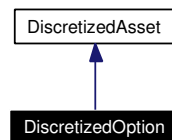
This method should initialize the asset values to an [Array](#) of the given size and with values depending on the particular asset.

Implements [DiscretizedAsset](#).

7.159 DiscretizedOption Class Reference

```
#include <ql/discretizedasset.hpp>
```

Inheritance diagram for DiscretizedOption:



7.159.1 Detailed Description

Discretized option on a given asset.

Warning:

it is advised that derived classes take care of creating and initializing themselves an instance of the underlying.

Public Member Functions

- **DiscretizedOption** (const boost::shared_ptr< [DiscretizedAsset](#) > &underlying, Exercise::Type exerciseType, const std::vector< [Time](#) > &exerciseTimes)
- void [reset](#) ([Size](#) size)
- std::vector< [Time](#) > [mandatoryTimes](#) () const

Protected Member Functions

- void [postAdjustValuesImpl](#) ()
- void [applyExerciseCondition](#) ()

Protected Attributes

- boost::shared_ptr< [DiscretizedAsset](#) > [underlying_](#)
- Exercise::Type [exerciseType_](#)
- std::vector< [Time](#) > [exerciseTimes_](#)

7.159.2 Member Function Documentation

7.159.2.1 void reset ([Size](#) size) [virtual]

This method should initialize the asset values to an [Array](#) of the given size and with values depending on the particular asset.

Implements [DiscretizedAsset](#).

7.159.2.2 `std::vector< Time > mandatoryTimes () const` [virtual]

This method returns the times at which the numerical method should stop while rolling back the asset. Typical examples include payment times, exercise times and such.

Note:

The returned values are not guaranteed to be sorted.

Reimplemented from [DiscretizedAsset](#).

7.159.2.3 `void postAdjustValuesImpl ()` [protected, virtual]

This method performs the actual post-adjustment

Reimplemented from [DiscretizedAsset](#).

7.160 Disposable Class Template Reference

```
#include <ql/disposable.hpp>
```

7.160.1 Detailed Description

template<class T> class QuantLib::Disposable< T >

generic disposable object with move semantics

This class can be used for returning a value by copy. It relies on the returned object exposing a `swap(T&)` method through which the copy constructor and assignment operator are implemented, thus resulting in actual move semantics. Typical use of this class is along the following lines:

```
Disposable<Foo> bar(Integer i) {  
    Foo f(i*2);  
    return f;  
}
```

Warning:

In order to avoid copies in code such as shown above, the conversion from `T` to `Disposable<T>` is destructive, i.e., it does **not** preserve the state of the original object. Therefore, it is necessary for the developer to avoid code such as

```
Disposable<Foo> bar(Foo& f) {  
    return f;  
}
```

which would likely render the passed object unusable. The correct way to obtain the desired behavior would be:

```
Disposable<Foo> bar(Foo& f) {  
    Foo temp = f;  
    return temp;  
}
```

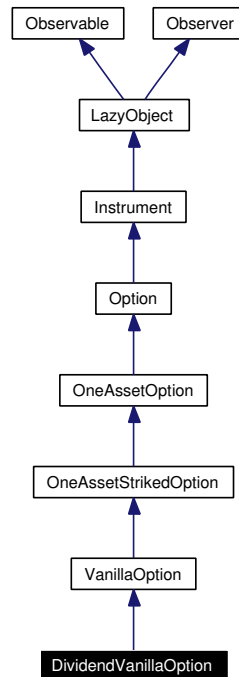
Public Member Functions

- **Disposable** (`T &t`)
- **Disposable** (`const Disposable< T > &t`)
- `Disposable< T > &operator=` (`const Disposable< T > &t`)

7.161 DividendVanillaOption Class Reference

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

Inheritance diagram for DividendVanillaOption:



7.161.1 Detailed Description

Single-asset vanilla option (no barriers) with discrete dividends.

Public Member Functions

- **DividendVanillaOption** (const boost::shared_ptr< [BlackScholesProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const std::vector< [Date](#) > ÷ndDates, const std::vector< [Real](#) > ÷nds, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())

Protected Member Functions

- void [setupArguments](#) ([Arguments](#) *) const

7.161.2 Member Function Documentation

7.161.2.1 void [setupArguments](#) ([Arguments](#) *) const [protected, virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetStrikedOption](#).

7.162 DividendVanillaOption::arguments Class Reference

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

7.162.1 Detailed Description

Arguments for dividend vanilla option calculation

Public Member Functions

- void **validate** () const

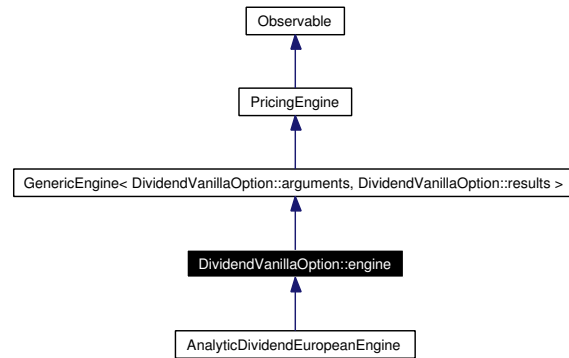
Public Attributes

- std::vector< [Date](#) > **dividendDates**
- std::vector< [Real](#) > **dividends**

7.163 DividendVanillaOption::engine Class Reference

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

Inheritance diagram for DividendVanillaOption::engine:



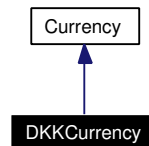
7.163.1 Detailed Description

Dividend vanilla option engine base class.

7.164 DKKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for DKKCurrency:



7.164.1 Detailed Description

Danish krone.

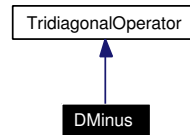
The ISO three-letter code is DKK; the numeric code is 208. It is divided in 100 øre.

ingroup currencies

7.165 DMinus Class Reference

```
#include <ql/FiniteDifferences/dminus.hpp>
```

Inheritance diagram for DMinus:



7.165.1 Detailed Description

D_- matricial representation

The differential operator D_- discretizes the first derivative with the first-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_i - u_{i-1}}{h} = D_- u_i$$

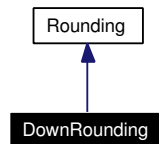
Public Member Functions

- **DMinus** ([Size](#) gridPoints, [Real](#) h)

7.166 DownRounding Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for DownRounding:



7.166.1 Detailed Description

Down-rounding.

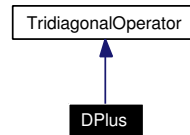
Public Member Functions

- **DownRounding** ([Integer](#) precision, [Integer](#) digit=5)

7.167 DPlus Class Reference

```
#include <ql/FiniteDifferences/dplus.hpp>
```

Inheritance diagram for DPlus:



7.167.1 Detailed Description

D_+ matricial representation

The differential operator D_+ discretizes the first derivative with the first-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_{i+1} - u_i}{h} = D_+ u_i$$

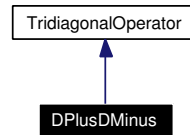
Public Member Functions

- DPlus (Size gridPoints, Real h)

7.168 DPlusDMinus Class Reference

```
#include <ql/FiniteDifferences/dplusdminus.hpp>
```

Inheritance diagram for DPlusDMinus:



7.168.1 Detailed Description

D_+D_- matricial representation

The differential operator D_+D_- discretizes the second derivative with the second-order formula

$$\frac{\partial^2 u_i}{\partial x^2} \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = D_+D_-u_i$$

Tests

the correctness of the returned values is tested by checking them against numerical calculations.

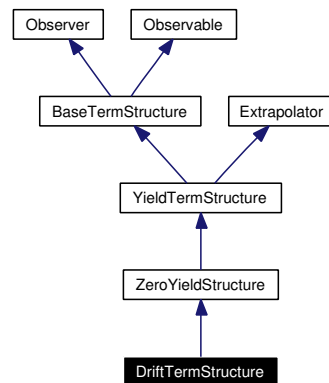
Public Member Functions

- **DPlusDMinus** ([Size](#) gridPoints, [Real](#) h)

7.169 DriftTermStructure Class Reference

```
#include <ql/TermStructures/drifttermstructure.hpp>
```

Inheritance diagram for DriftTermStructure:



7.169.1 Detailed Description

Drift term structure.

Drift term structure for modelling the common drift term: $\text{riskFreeRate} - \text{dividendYield} - 0.5 \cdot \text{vol} \cdot \text{vol}$

Note:

This term structure will remain linked to the original structures, i.e., any changes in the latters will be reflected in this structure as well.

Public Member Functions

- **DriftTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, const [Handle](#)< [BlackVolTermStructure](#) > &blackVolTS)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- const [Date](#) & [todaysDate](#) () const
today's date
- const [Date](#) & [referenceDate](#) () const
the reference date, i.e., the date at which discount = 1
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return rates

Protected Member Functions

- [Rate zeroYieldImpl](#) ([Time](#)) const
returns the discount factor as seen from the evaluation date

7.169.2 Member Function Documentation

7.169.2.1 const [Date](#) & todaysDate () const [virtual]

today's date

Deprecated

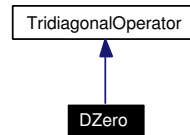
use [Settings::instance\(\).evaluationDate\(\)](#).

Reimplemented from [BaseTermStructure](#).

7.170 DZero Class Reference

```
#include <ql/FiniteDifferences/dzero.hpp>
```

Inheritance diagram for DZero:



7.170.1 Detailed Description

D_0 matricial representation

The differential operator D_0 discretizes the first derivative with the second-order formula

$$\frac{\partial u_i}{\partial x} \approx \frac{u_{i+1} - u_{i-1}}{2h} = D_0 u_i$$

Tests

the correctness of the returned values is tested by checking them against numerical calculations.

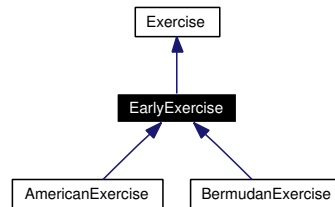
Public Member Functions

- **DZero** ([Size](#) gridPoints, [Real](#) h)

7.171 EarlyExercise Class Reference

```
#include <ql/exercise.hpp>
```

Inheritance diagram for EarlyExercise:



7.171.1 Detailed Description

Early-exercise base class.

The payoff can be at exercise (default case) or at expiry

Todo

derive a plain American `Exercise` class (no `earliestDate`, no `payoffAtExpiry`)

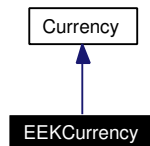
Public Member Functions

- **EarlyExercise** (bool `payoffAtExpiry`=false)
- bool **payoffAtExpiry** () const

7.172 EEKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for EEKCurrency:



7.172.1 Detailed Description

Estonian kroon.

The ISO three-letter code is `EEK`; the numeric code is 233. It is divided in 100 senti.

ingroup currencies

7.173 EndCriteria Class Reference

```
#include <ql/Optimization/criteria.hpp>
```

7.173.1 Detailed Description

Criteria to end optimization process.

- stationary point
 - stationary gradient
 - maximum number of iterations

Public Types

- enum Type { none, maxIter, statPt, statGd }

Public Member Functions

- [EndCriteria](#) ()
default constructor
- [EndCriteria](#) ([Size](#) maxIteration, [Real](#) epsilon)
initialization constructor
- void **setPositiveOptimization** ()
- bool **checkIterationNumber** ([Size](#) iteration)
- bool **checkStationaryValue** ([Real](#) fold, [Real](#) fnew)
- bool **checkAccuracyValue** ([Real](#) f)
- bool **checkStationaryGradientNorm** ([Real](#) normDiff)
- bool **checkAccuracyGradientNorm** ([Real](#) norm)
- bool **operator()** ([Size](#) iteration, [Real](#) fold, [Real](#) normgold, [Real](#) fnew, [Real](#) normgnew, [Real](#))
test if the number of iteration is not too big and if we don't
- Type **criteria** () const
return the end criteria type

Protected Attributes

- [Size](#) maxIteration_
Maximum number of iterations.
- [Real](#) functionEpsilon_
function and gradient epsilons
- [Real](#) gradientEpsilon_
function and gradient epsilons

- [Size maxIterStatPt_](#)

Maximun number of iterations in stationary state.

- [Size statState_](#)

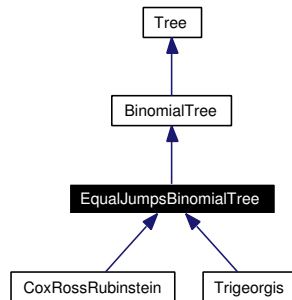
Maximun number of iterations in stationary state.

- Type **endCriteria_**
- bool **positiveOptimization_**

7.174 EqualJumpsBinomialTree Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for EqualJumpsBinomialTree:



7.174.1 Detailed Description

Base class for equal jumps binomial tree.

Public Member Functions

- `EqualJumpsBinomialTree` (const boost::shared_ptr< [StochasticProcess](#) > &process, [Time](#) end, [Size](#) steps)
- `Real underlying` ([Size](#) i, [Size](#) index) const
- `Real probability` ([Size](#), [Size](#), [Size](#) branch) const

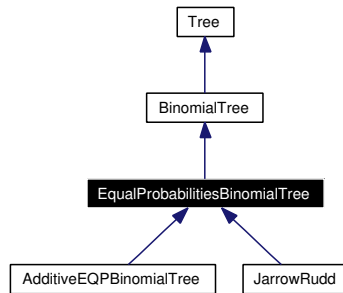
Protected Attributes

- `Real dx_`
- `Real pu_`
- `Real pd_`

7.175 EqualProbabilitiesBinomialTree Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for EqualProbabilitiesBinomialTree:



7.175.1 Detailed Description

Base class for equal probabilities binomial tree.

Public Member Functions

- `EqualProbabilitiesBinomialTree` (const boost::shared_ptr< [StochasticProcess](#) > &process, [Time](#) end, [Size](#) steps)
- `Real underlying` ([Size](#) i, [Size](#) index) const
- `Real probability` ([Size](#), [Size](#), [Size](#)) const

Protected Attributes

- `Real up_`

7.176 Error Class Reference

```
#include <ql/errors.hpp>
```

7.176.1 Detailed Description

Base error class.

Public Member Functions

- [Error](#) (const std::string &file, long line, const std::string &function, const std::string &message="")
- [~Error](#) () throw ()
- const char * [what](#) () const throw ()
returns the error message.

7.176.2 Constructor & Destructor Documentation

7.176.2.1 [Error](#) (const std::string &file, long line, const std::string &function, const std::string &message = "")

The explicit use of this constructor is not advised. Use the `QL_FAIL` macro instead.

7.176.2.2 [~Error](#) () throw ()

the automatically generated destructor would not have the throw specifier.

7.177 ErrorFunction Class Reference

```
#include <ql/Math/errorfunction.hpp>
```

7.177.1 Detailed Description

Error function

formula here ... Used to calculate the cumulative normal distribution function

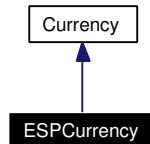
Public Member Functions

- [Real](#) operator() ([Real](#) x) const

7.178 ESPCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ESPCurrency:



7.178.1 Detailed Description

Spanish peseta.

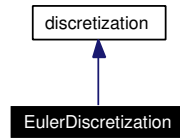
The ISO three-letter code is ESP; the numeric code is 724. It is divided in 100 centimos.

ingroup currencies

7.179 EulerDiscretization Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for EulerDiscretization:



7.179.1 Detailed Description

Euler discretization for stochastic processes.

Public Member Functions

- [Real expectation](#) (const [StochasticProcess](#) &, [Time](#) t0, [Real](#) x0, [Time](#) dt) const
- [Real variance](#) (const [StochasticProcess](#) &, [Time](#) t0, [Real](#) x0, [Time](#) dt) const

7.179.2 Member Function Documentation

7.179.2.1 [Real expectation](#) (const [StochasticProcess](#) &, [Time](#) t0, [Real](#) x0, [Time](#) dt) const [virtual]

Returns an approximation of the expected value defined as $x_0 + \mu(t_0, x_0)\Delta t$.

Implements [StochasticProcess::discretization](#).

7.179.2.2 [Real variance](#) (const [StochasticProcess](#) &, [Time](#) t0, [Real](#) x0, [Time](#) dt) const [virtual]

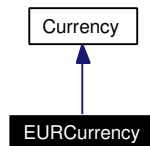
Returns an approximation of the variance defined as $\sigma(t_0, x_0)^2\Delta t$.

Implements [StochasticProcess::discretization](#).

7.180 EURCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for EURCurrency:



7.180.1 Detailed Description

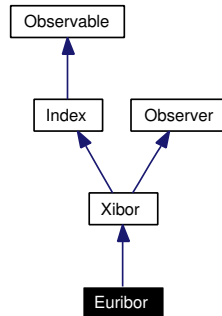
European Euro.

The ISO three-letter code is EUR; the numeric code is 978. It is divided into 100 cents.

7.181 Euribor Class Reference

```
#include <ql/Indexes/euribor.hpp>
```

Inheritance diagram for Euribor:



7.181.1 Detailed Description

Euribor index

Public Member Functions

- **Euribor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

7.182 EuroFormatter Class Reference

```
#include <ql/dataformatters.hpp>
```

7.182.1 Detailed Description

Formats amounts in Euro for output.

Formatting follows Euro convention (x,xxx,xxx.xx)

Deprecated

use [MoneyFormatter](#) instead

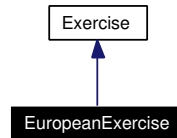
Static Public Member Functions

- `std::string toString` ([Decimal](#) amount)

7.183 EuropeanExercise Class Reference

```
#include <ql/exercise.hpp>
```

Inheritance diagram for EuropeanExercise:



7.183.1 Detailed Description

European exercise.

A European option can only be exercised at one (expiry) date.

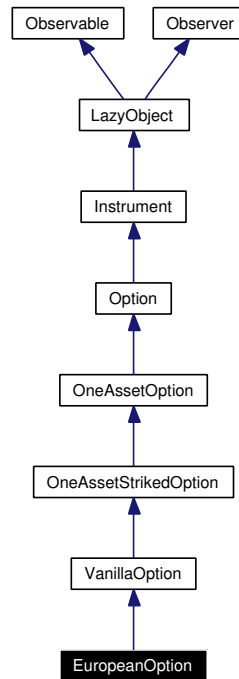
Public Member Functions

- `EuropeanExercise` ([Date](#) date)

7.184 EuropeanOption Class Reference

```
#include <ql/Instruments/europeanoption.hpp>
```

Inheritance diagram for EuropeanOption:



7.184.1 Detailed Description

European option on a single asset.

Public Member Functions

- **EuropeanOption** (const boost::shared_ptr< [BlackScholesProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())

7.185 ExchangeRate Class Reference

```
#include <ql/exchangerate.hpp>
```

7.185.1 Detailed Description

exchange rate between two currencies

Tests

application of direct and derived exchange rate is tested against calculations.

Utility methods

- [Money exchange](#) (const [Money](#) &amount) const
apply the exchange rate to a cash amount
- [ExchangeRate chain](#) (const [ExchangeRate](#) &r1, const [ExchangeRate](#) &r2)
chain two exchange rates

Public Types

- enum [Type](#) { [Direct](#), [Derived](#) }

Public Member Functions

Constructors

- [ExchangeRate](#) (const [Currency](#) &source, const [Currency](#) &target, [Decimal](#) rate)

Inspectors

- const [Currency](#) & [source](#) () const
the source currency.
- const [Currency](#) & [target](#) () const
the target currency.
- [Type](#) [type](#) () const
the type
- [Decimal](#) [rate](#) () const
the exchange rate (when available)

7.185.2 Member Enumeration Documentation

7.185.2.1 enum [Type](#)

Enumeration values:

Direct given directly by the user

Derived derived from exchange rates between other currencies

7.185.3 Constructor & Destructor Documentation

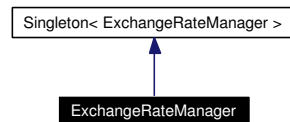
7.185.3.1 [ExchangeRate](#) (const [Currency](#) & *source*, const [Currency](#) & *target*, [Decimal](#) *rate*)

the rate r is given with the convention that a unit of the source is worth r units of the target.

7.186 ExchangeRateManager Class Reference

```
#include <ql/Currencies/exchangeratemanager.hpp>
```

Inheritance diagram for ExchangeRateManager:



7.186.1 Detailed Description

exchange-rate repository

Tests

lookup of direct, triangulated, and derived exchange rates is tested.

Public Member Functions

- void **add** (const [ExchangeRate](#) &, const [Date](#) &startDate=[Date::minDate\(\)](#), const [Date](#) &endDate=[Date::maxDate\(\)](#))
Add an exchange rate.
- [ExchangeRate](#) **lookup** (const [Currency](#) &source, const [Currency](#) &target, [Date](#) date=[Date\(\)](#), [ExchangeRate::Type](#) type=[ExchangeRate::Derived](#)) const
- void **clear** ()
remove the added exchange rates

Friends

- class [Singleton](#)<[ExchangeRateManager](#)>

7.186.2 Member Function Documentation

7.186.2.1 void **add** (const [ExchangeRate](#) &, const [Date](#) & *startDate* = [Date::minDate\(\)](#), const [Date](#) & *endDate* = [Date::maxDate\(\)](#))

Add an exchange rate.

The given rate is valid between the given dates.

Note:

If two rates are given between the same currencies and with overlapping date ranges, the latest one added takes precedence during lookup.

7.186.2.2 **ExchangeRate** lookup (const **Currency** & *source*, const **Currency** & *target*, **Date** *date* = **Date**(), **ExchangeRate::Type** *type* = ExchangeRate::Derived) const

Lookup the exchange rate between two currencies at a given date. If the given type is Direct, only direct exchange rates will be returned if available; if Derived, direct rates are still preferred but derived rates are allowed.

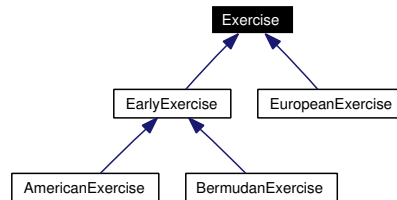
Warning:

if two or more exchange-rate chains are possible which allow to specify a requested rate, it is unspecified which one is returned.

7.187 Exercise Class Reference

```
#include <ql/exercise.hpp>
```

Inheritance diagram for Exercise:



7.187.1 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`

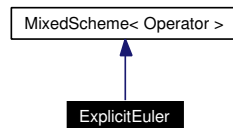
Protected Attributes

- `std::vector< Date > dates_`
- `Type type_`

7.188 ExplicitEuler Class Template Reference

```
#include <ql/FiniteDifferences/expliciteuler.hpp>
```

Inheritance diagram for ExplicitEuler:



7.188.1 Detailed Description

```
template<class Operator> class QuantLib::ExplicitEuler< Operator >
```

Forward Euler scheme for finite difference methods.

See sect. [Finite-differences framework](#) for details on the method.

In this implementation, the passed operator must be derived from either `TimeConstantOperator` or `TimeDependentOperator`. Also, it must implement at least the following interface:

```
typedef ... 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*(Real, const Operator&);
Operator operator-(const Operator&, const Operator&);
```

Todo

add Richardson extrapolation

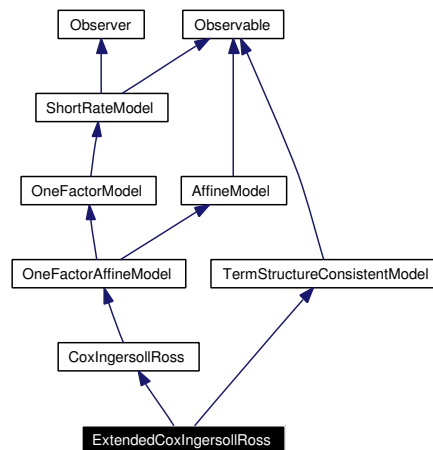
Friends

- class `FiniteDifferenceModel<ExplicitEuler<Operator> >`

7.189 ExtendedCoxIngersollRoss Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp>
```

Inheritance diagram for ExtendedCoxIngersollRoss:



7.189.1 Detailed Description

Extended Cox-Ingersoll-Ross model class.

This class implements the extended Cox-Ingersoll-Ross model defined by

$$dr_t = (\theta(t) - \alpha r_t)dt + \sqrt{r_t} \sigma dW_t.$$

Bug

this class was not tested enough to guarantee its functionality.

Public Member Functions

- **ExtendedCoxIngersollRoss** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) theta=0.1, [Real](#) k=0.1, [Real](#) sigma=0.1, [Real](#) x0=0.05)
- [boost::shared_ptr](#)< [Lattice](#) > [tree](#) (const [TimeGrid](#) &grid) const
Return by default a trinomial recombining tree.
- [boost::shared_ptr](#)< [ShortRateDynamics](#) > [dynamics](#) () const
returns the short-rate dynamics
- [Real](#) [discountBondOption](#) ([Option::Type](#) type, [Real](#) strike, [Time](#) maturity, [Time](#) bond-Maturity) const

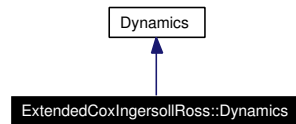
Protected Member Functions

- void [generateArguments](#) ()
- [Real](#) [A](#) ([Time](#) t, [Time](#) T) const

7.190 ExtendedCoxIngersollRoss::Dynamics Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp>
```

Inheritance diagram for ExtendedCoxIngersollRoss::Dynamics:



7.190.1 Detailed Description

Short-rate dynamics in the extended Cox-Ingersoll-Ross model.

The short-rate is here

$$r_t = \varphi(t) + y_t^2$$

where $\varphi(t)$ is the deterministic time-dependent parameter used for term-structure fitting and y_t is the state variable, the square-root of a standard CIR process.

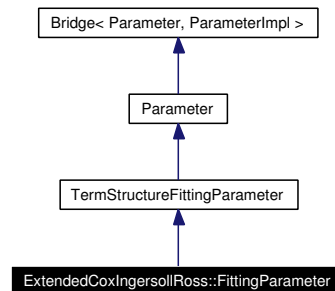
Public Member Functions

- **Dynamics** (const [Parameter](#) &phi, [Real](#) theta, [Real](#) k, [Real](#) sigma, [Real](#) x0)
- virtual [Real](#) **variable** ([Time](#) t, [Rate](#) r) const
- virtual [Real](#) **shortRate** ([Time](#) t, [Real](#) y) const

7.191 ExtendedCoxIngersollRoss::FittingParameter Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp>
```

Inheritance diagram for ExtendedCoxIngersollRoss::FittingParameter:



7.191.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) - \frac{2k\theta(e^{th} - 1)}{2h + (k + h)(e^{th} - 1)} - \frac{4x_0h^2e^{th}}{(2h + (k + h)(e^{th} - 1))^1},$$

where $f(t)$ is the instantaneous forward rate at t and $h = \sqrt{k^2 + 2\sigma^2}$.

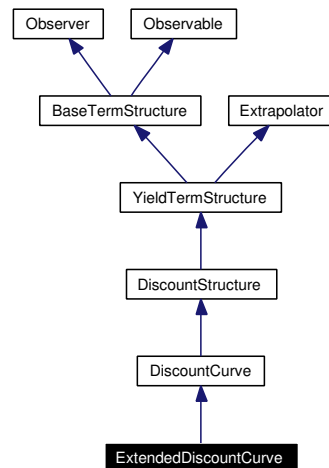
Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) theta, [Real](#) k, [Real](#) sigma, [Real](#) x0)

7.192 ExtendedDiscountCurve Class Reference

```
#include <ql/TermStructures/extendeddiscountcurve.hpp>
```

Inheritance diagram for ExtendedDiscountCurve:



7.192.1 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 [BusinessDayConvention](#) conv, const [DayCounter](#) &dayCounter)
- **ExtendedDiscountCurve** (const std::vector< [Date](#) > &dates, const std::vector< [DiscountFactor](#) > &dfs, const [Calendar](#) &calendar, const [BusinessDayConvention](#) conv, const [DayCounter](#) &dayCounter)
- [Calendar](#) **calendar** () const
the calendar used for reference date calculation
- [BusinessDayConvention](#) **businessDayConvention** () const
- void **update** ()

Protected Member Functions

- [Rate](#) **compoundForwardImpl** ([Time](#), [Integer](#)) const
- void **calibrateNodes** () const
- boost::shared_ptr< [YieldTermStructure](#) > **reversebootstrap** ([Integer](#)) const
- boost::shared_ptr< [YieldTermStructure](#) > **forwardCurve** ([Integer](#)) const

7.192.2 Constructor & Destructor Documentation

7.192.2.1 **ExtendedDiscountCurve** (const **Date** & *today'sDate*, const std::vector< **Date** > & *dates*, const std::vector< **DiscountFactor** > & *dfs*, const **Calendar** & *calendar*, const **BusinessDayConvention** *conv*, const **DayCounter** & *dayCounter*)

Deprecated

use the constructor without today's date

7.192.3 Member Function Documentation

7.192.3.1 **void update ()** [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [BaseTermStructure](#).

7.192.3.2 **Rate compoundForwardImpl (Time, Integer) const** [protected, virtual]

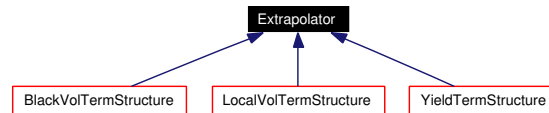
Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Reimplemented from [DiscountStructure](#).

7.193 Extrapolator Class Reference

```
#include <ql/Math/extrapolation.hpp>
```

Inheritance diagram for Extrapolator:



7.193.1 Detailed Description

base class for classes possibly allowing extrapolation

Public Member Functions

modifiers

- void [enableExtrapolation](#) ()
enable extrapolation in subsequent calls
- void [disableExtrapolation](#) ()
disable extrapolation in subsequent calls

inspectors

- bool [allowsExtrapolation](#) () const
tells whether extrapolation is enabled

7.194 Factorial Class Reference

```
#include <ql/Math/factorial.hpp>
```

7.194.1 Detailed Description

Factorial numbers calculator

Tests

the correctness of the returned value is tested by checking it against numerical calculations.

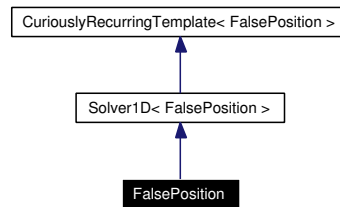
Static Public Member Functions

- [Real](#) `get` ([Natural](#) n)
- [Real](#) `ln` ([Natural](#) n)

7.195 FalsePosition Class Reference

```
#include <ql/Solvers1D/falseposition.hpp>
```

Inheritance diagram for FalsePosition:



7.195.1 Detailed Description

False position 1-D solver.

Tests

the correctness of the returned values is tested by checking them against known good results.

Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

7.196 FaureRsg Class Reference

```
#include <ql/RandomNumbers/faurersg.hpp>
```

7.196.1 Detailed Description

Faure low-discrepancy sequence generator.

It is based on existing Fortran and C algorithms to calculate pascal matrix and gray transforms.

1. E. Thiernard Economic generation of low-discrepancy sequences with a b-ary gray code.
2. Algorithms 659, 647. <http://www.netlib.org/toms/647>,
<http://www.netlib.org/toms/659>

Tests

the correctness of the returned values is tested by reproducing known good values.

Public Types

- typedef [Sample](#)< [Array](#) > **sample_type**

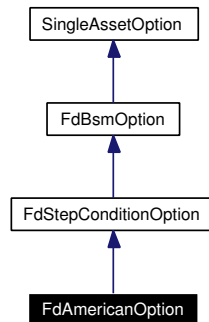
Public Member Functions

- **FaureRsg** ([Size](#) dimensionality)
- const std::vector< long int > & **nextIntSequence** () const
- const std::vector< long int > & **lastIntSequence** () const
- const [sample_type](#) & **nextSequence** () const
- const [sample_type](#) & **lastSequence** () const
- [Size](#) **dimension** () const

7.197 FdAmericanOption Class Reference

```
#include <ql/Pricers/fdamericanoption.hpp>
```

Inheritance diagram for FdAmericanOption:



7.197.1 Detailed Description

American option.

Public Member Functions

- **FdAmericanOption** (Option::Type type, [Real](#) underlying, [Real](#) strike, [Spread](#) dividend-Yield, [Rate](#) riskFreeRate, [Time](#) residualTime, [Volatility](#) volatility, [Size](#) timeSteps=100, [Size](#) gridPoints=100)
- void **initializeStepCondition** () const
- boost::shared_ptr< [SingleAssetOption](#) > **clone** () const

7.198 FdBermudanOption Class Reference

```
#include <ql/Pricers/fdbermudanoption.hpp>
```

7.198.1 Detailed Description

Bermudan option.

Public Member Functions

- **FdBermudanOption** (Option::Type type, [Real](#) underlying, [Real](#) strike, [Spread](#) dividend-Yield, [Rate](#) riskFreeRate, [Time](#) residualTime, [Volatility](#) volatility, const std::vector< [Time](#) > &dates=std::vector< [Time](#) >(), [Size](#) timeSteps=100, [Size](#) gridPoints=100)
- boost::shared_ptr< [SingleAssetOption](#) > **clone** () const

Protected Member Functions

- void **initializeStepCondition** () const
- void **executeIntermediateStep** ([Size](#)) const

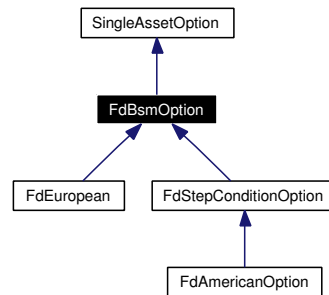
Protected Attributes

- [Real](#) extraTermInBermudan

7.199 FdBsmOption Class Reference

```
#include <ql/Pricers/fdbsmoption.hpp>
```

Inheritance diagram for FdBsmOption:



7.199.1 Detailed Description

Black-Scholes-Merton option priced numerically.

Public Member Functions

- **FdBsmOption** (Option::Type type, [Real](#) underlying, [Real](#) strike, [Spread](#) dividendYield, [Rate](#) riskFreeRate, [Time](#) residualTime, [Volatility](#) volatility, [Size](#) gridPoints)
- virtual void **calculate** () const =0
- [Real](#) **value** () const
- [Real](#) **delta** () const
- [Real](#) **gamma** () const
- const [Array](#) & **getGrid** () const

Protected Types

- typedef [BoundaryCondition](#)< [TridiagonalOperator](#) > **BoundaryCondition**

Protected Member Functions

- virtual void **setGridLimits** ([Real](#) center, [Real](#) timeDelay) const
- virtual void **initializeGrid** () const
- virtual void **initializeInitialCondition** () const
- virtual void **initializeOperator** () const

Protected Attributes

- [Size](#) gridPoints_
- [Real](#) value_
- [Real](#) delta_
- [Real](#) gamma_

- [Array](#) grid_
- [BSMOperator](#) finiteDifferenceOperator_
- [Array](#) intrinsicValues_
- std::vector< boost::shared_ptr< [BoundaryCondition](#) > > BCs_
- [Real](#) sMin_
- [Real](#) center_
- [Real](#) sMax_

7.200 FdDividendAmericanOption Class Reference

```
#include <ql/Pricers/fddividendamericanoption.hpp>
```

7.200.1 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, [Real](#) underlying, [Real](#) strike, [Spread](#) dividendYield, [Rate](#) riskFreeRate, [Time](#) residualTime, [Volatility](#) volatility, const std::vector< [Real](#) > ÷nds=std::vector< [Real](#) >(), const std::vector< [Time](#) > &exdivdates=std::vector< [Time](#) >(), [Size](#) timeSteps=100, [Size](#) gridPoints=100)
- boost::shared_ptr< [SingleAssetOption](#) > **clone** () const

7.201 FdDividendShoutOption Class Reference

```
#include <ql/Pricers/fddividendshoutoption.hpp>
```

7.201.1 Detailed Description

Shout option with dividends.

Public Member Functions

- **FdDividendShoutOption** (Option::Type type, [Real](#) underlying, [Real](#) strike, [Spread](#) dividendYield, [Rate](#) riskFreeRate, [Time](#) residualTime, [Volatility](#) volatility, const std::vector< [Real](#) > ÷nds=std::vector< [Real](#) >(), const std::vector< [Time](#) > &exdivdates=std::vector< [Time](#) >(), [Size](#) timeSteps=100, [Size](#) gridPoints=100)
- boost::shared_ptr< [SingleAssetOption](#) > **clone** () const
- [Real](#) **dividendRho** () const

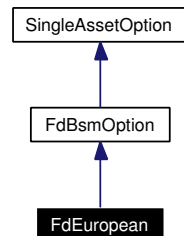
Protected Member Functions

- void **initializeStepCondition** () const

7.202 FdEuropean Class Reference

```
#include <ql/Pricers/fdeuropean.hpp>
```

Inheritance diagram for FdEuropean:



7.202.1 Detailed Description

Example of European option calculated using finite differences.

Public Member Functions

- **FdEuropean** (Option::Type type, [Real](#) underlying, [Real](#) strike, [Spread](#) dividendYield, [Rate](#) riskFreeRate, [Time](#) residualTime, [Volatility](#) volatility, [Size](#) timeSteps=200, [Size](#) gridPoints=800)
- const [Array](#) & **getPrices** () const
- boost::shared_ptr< [SingleAssetOption](#) > **clone** () const

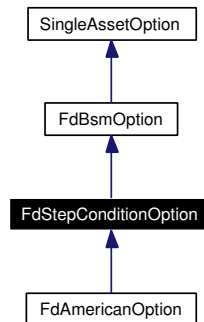
Protected Member Functions

- void **calculate** () const

7.203 FdStepConditionOption Class Reference

```
#include <ql/Pricers/fdstepconditionoption.hpp>
```

Inheritance diagram for FdStepConditionOption:



7.203.1 Detailed Description

option executing additional code at each time step

Protected Member Functions

- **FdStepConditionOption** (Option::Type type, [Real](#) underlying, [Real](#) strike, [Spread](#) dividendYield, [Rate](#) riskFreeRate, [Time](#) residualTime, [Volatility](#) volatility, [Size](#) timeSteps, [Size](#) gridPoints)
- void **calculate** () const
- virtual void **initializeStepCondition** () const =0

Protected Attributes

- boost::shared_ptr< [StandardStepCondition](#) > **stepCondition_**
- [Size](#) **timeSteps_**

7.204 `filtering_iterator` Class Template Reference

```
#include <ql/Utilities/filteringiterator.hpp>
```

7.204.1 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.

Deprecated

use `boost::filter_iterator` instead

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

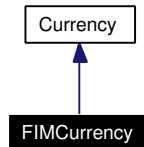
(Note that these are not member functions.)

- `filtering_iterator`< `Iterator`, `UnaryPredicate` > `make_filtering_iterator` (`Iterator` it, `UnaryPredicate` p, `Iterator` beforeBegin, `Iterator` end)
helper function to create filtering iterators

7.205 FIMCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for FIMCurrency:



7.205.1 Detailed Description

Finnish markka.

The ISO three-letter code is FIM; the numeric code is 246. It is divided in 100 penniä.

ingroup currencies

7.206 FiniteDifferenceModel Class Template Reference

```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
```

7.206.1 Detailed Description

```
template<class Evolver> class QuantLib::FiniteDifferenceModel< Evolver >
```

Generic finite difference model.

See sect. [Finite-differences framework](#)

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< boost::shared_ptr< [bcType](#) > > &bcs, const std::vector< [Time](#) > &stoppingTimes=std::vector< [Time](#) >())
- **FiniteDifferenceModel** (const Evolver &evolver, const std::vector< [Time](#) > &stoppingTimes=std::vector< [Time](#) >())
- const Evolver & **evolver** () const
- void **rollback** (arrayType &a, [Time](#) from, [Time](#) to, [Size](#) steps, const boost::shared_ptr< [conditionType](#) > &condition=boost::shared_ptr< [conditionType](#) >())

7.206.2 Member Function Documentation

7.206.2.1 void rollback (arrayType & a, [Time](#) from, [Time](#) to, [Size](#) steps, const boost::shared_ptr< [conditionType](#) > & condition = boost::shared_ptr<[conditionType](#)>())

solves the problem between the given times, possibly applying a condition at every step.

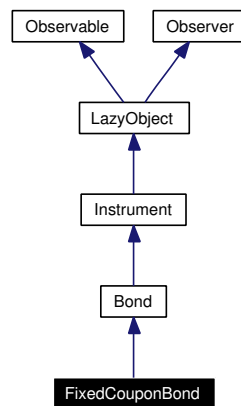
Warning:

being this a rollback, from must be a later time than to.

7.207 FixedCouponBond Class Reference

```
#include <ql/Instruments/fixedcouponbond.hpp>
```

Inheritance diagram for FixedCouponBond:



7.207.1 Detailed Description

fixed-coupon bond

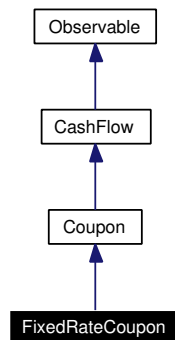
Public Member Functions

- **FixedCouponBond** (const [Date](#) &issueDate, const [Date](#) &datedDate, const [Date](#) &maturityDate, [Integer](#) settlementDays, [Rate](#) coupon, [Frequency](#) couponFrequency, const [DayCounter](#) &dayCounter, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention=Following, [Real](#) redemption=100.0)

7.208 FixedRateCoupon Class Reference

```
#include <ql/CashFlows/fixedratecoupon.hpp>
```

Inheritance diagram for FixedRateCoupon:



7.208.1 Detailed Description

Coupon paying a fixed interest rate

Public Member Functions

- **FixedRateCoupon** ([Real](#) nominal, const [Date](#) &paymentDate, [Rate](#) rate, const [DayCounter](#) &dayCounter, const [Date](#) &startDate, const [Date](#) &endDate, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)())

CashFlow interface

- [Real](#) **amount** () const
returns the amount of the cash flow

Coupon interface

- [Rate](#) **rate** () const
accrued rate
- [DayCounter](#) **dayCounter** () const
day counter for accrual calculation
- [Real](#) **accruedAmount** (const [Date](#) &) const
accrued amount at the given date

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

7.208.2 Member Function Documentation

7.208.2.1 [Real](#) amount () const [virtual]

returns the amount of the cash flow

Note:

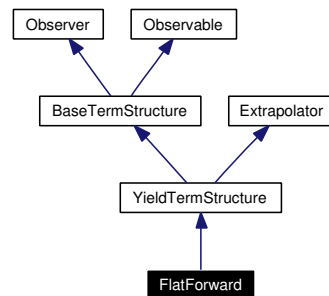
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

7.209 FlatForward Class Reference

```
#include <ql/TermStructures/flatforward.hpp>
```

Inheritance diagram for FlatForward:



7.209.1 Detailed Description

Flat interest-rate curve.

Public Member Functions

- **FlatForward** (const [Date](#) &todayDate, const [Date](#) &referenceDate, [Rate](#) forward, const [DayCounter](#) &dayCounter)
- **FlatForward** (const [Date](#) &todayDate, const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &forward, const [DayCounter](#) &dayCounter)
- **FlatForward** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &forward, const [DayCounter](#) &dayCounter)
- **FlatForward** (const [Date](#) &referenceDate, [Rate](#) forward, const [DayCounter](#) &dayCounter)
- **FlatForward** ([Integer](#) settlementDays, const [Calendar](#) &calendar, const [Handle](#)< [Quote](#) > &forward, const [DayCounter](#) &dayCounter)
- **FlatForward** ([Integer](#) settlementDays, const [Calendar](#) &calendar, [Rate](#) forward, const [DayCounter](#) &dayCounter)
- **DayCounter** dayCounter () const
the day counter used for date/time conversion
- **Date** maxDate () const
the latest date for which the curve can return rates

Protected Member Functions

- **Rate** zeroYieldImpl ([Time](#)) const
zero-yield calculation
- **DiscountFactor** discountImpl ([Time](#)) const
discount calculation
- **Rate** forwardImpl ([Time](#)) const

instantaneous forward-rate calculation

- [Rate compoundForwardImpl](#) ([Time](#) t, [Integer](#) compFreq) const
compound forward-rate calculation

7.209.2 Constructor & Destructor Documentation

7.209.2.1 [FlatForward](#) (const [Date](#) & *today'sDate*, const [Date](#) & *referenceDate*, [Rate](#) *forward*, const [DayCounter](#) & *dayCounter*)

Deprecated

use one of the non-deprecated constructors.

7.209.2.2 [FlatForward](#) (const [Date](#) & *today'sDate*, const [Date](#) & *referenceDate*, const [Handle](#)<[Quote](#)> & *forward*, const [DayCounter](#) & *dayCounter*)

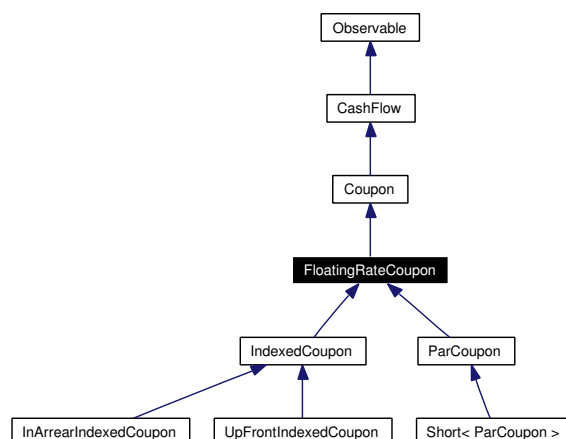
Deprecated

use one of the non-deprecated constructors.

7.210 FloatingRateCoupon Class Reference

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

Inheritance diagram for FloatingRateCoupon:



7.210.1 Detailed Description

Coupon paying a variable rate

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **FloatingRateCoupon** ([Real](#) nominal, const [Date](#) &paymentDate, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)())

Coupon interface

- [Rate](#) **rate** () const
accrued rate
- [Real](#) **accruedAmount** (const [Date](#) &) const
accrued amount at the given date

Inspectors

- [Integer](#) **fixingDays** () const
fixing days
- virtual [Spread](#) **spread** () const
spread paid over the fixing of the underlying index

- virtual [Rate indexFixing](#) () const =0
fixing of the underlying index
- virtual [Rate fixing](#) () const =0
- virtual [Date fixingDate](#) () const =0
fixing date

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Member Functions

- virtual [Rate convexityAdjustment](#) ([Rate](#) fixing) const
convexity adjustment for the given index fixing

Protected Attributes

- [Integer](#) fixingDays_
- [Spread](#) spread_

7.210.2 Member Function Documentation

7.210.2.1 virtual [Rate](#) fixing () const [pure virtual]

Deprecated

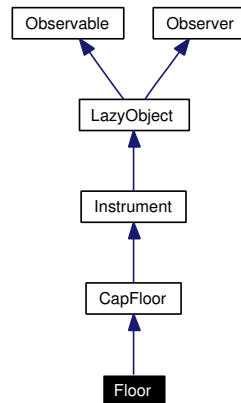
use [rate\(\)](#) instead

Implemented in [IndexedCoupon](#), and [ParCoupon](#).

7.211 Floor Class Reference

```
#include <ql/Instruments/capfloor.hpp>
```

Inheritance diagram for Floor:



7.211.1 Detailed Description

Concrete floor class.

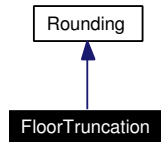
Public Member Functions

- **Floor** (const std::vector< boost::shared_ptr< [CashFlow](#) > > &floatingLeg, const std::vector< [Rate](#) > &exerciseRates, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared_ptr< [PricingEngine](#) > &engine)

7.212 FloorTruncation Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for FloorTruncation:



7.212.1 Detailed Description

[Floor](#) truncation.

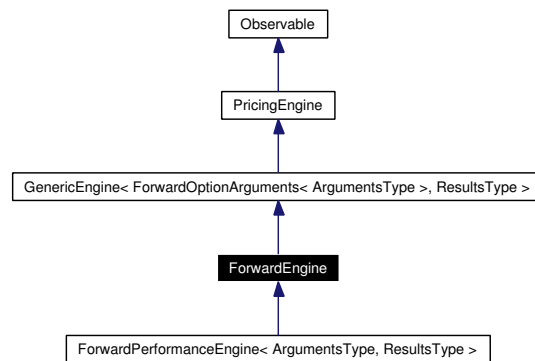
Public Member Functions

- `FloorTruncation` ([Integer](#) precision, [Integer](#) digit=5)

7.213 ForwardEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

Inheritance diagram for ForwardEngine:



7.213.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::ForwardEngine<
ArgumentsType, ResultsType >
```

Forward engine base class.

Tests

- the correctness of the returned value is tested by reproducing results available in literature.
- the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **ForwardEngine** (const boost::shared_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > &)
- void **setOriginalArguments** () const
- void **calculate** () const
- void **getOriginalResults** () const

Protected Attributes

- boost::shared_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

7.214 ForwardOptionArguments Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

7.214.1 Detailed Description

```
template<class ArgumentsType> class QuantLib::ForwardOptionArguments< Arguments-  
Type >
```

Arguments for forward (strike-resetting) option calculation

Public Member Functions

- `void validate () const`

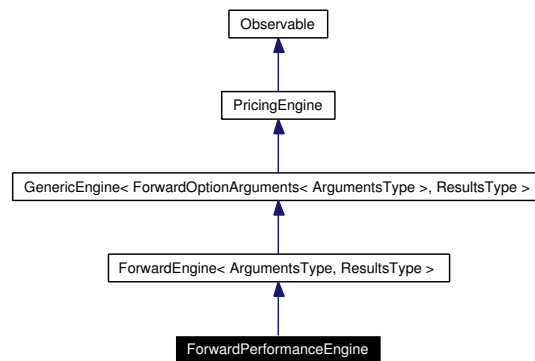
Public Attributes

- [Real](#) `moneyness`
- [Date](#) `resetDate`

7.215 ForwardPerformanceEngine Class Template Reference

```
#include <ql/PricingEngines/Forward/forwardperformanceengine.hpp>
```

Inheritance diagram for ForwardPerformanceEngine:



7.215.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::ForwardPerformance-  
Engine< ArgumentsType, ResultsType >
```

Forward performance engine.

Tests

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

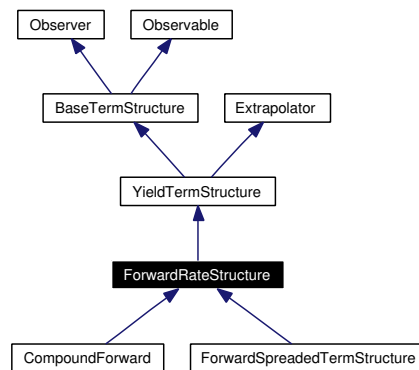
Public Member Functions

- **ForwardPerformanceEngine** (const boost::shared_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > > &)
- void **calculate** () const
- void **getOriginalResults** () const

7.216 ForwardRateStructure Class Reference

```
#include <ql/TermStructures/forwardstructure.hpp>
```

Inheritance diagram for ForwardRateStructure:



7.216.1 Detailed Description

Forward rate term structure.

This abstract class acts as an adapter to TermStructure allowing the programmer to implement only the `forwardImpl(const Date&, bool)` method in derived classes. Zero yields and discounts are calculated from forwards.

Rates are assumed to be annual continuous compounding.

Public Member Functions

Constructors

See the *BaseTermStructure* documentation for issues regarding constructors.

- [ForwardRateStructure](#) (const [Date](#) &todayDate, const [Date](#) &referenceDate)
- [ForwardRateStructure](#) ()
- [ForwardRateStructure](#) (const [Date](#) &referenceDate)
- [ForwardRateStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)

Protected Member Functions

YieldTermStructure implementation

- [DiscountFactor](#) [discountImpl](#) ([Time](#)) const
- virtual [Rate](#) [forwardImpl](#) ([Time](#)) const =0
instantaneous forward-rate calculation
- virtual [Rate](#) [zeroYieldImpl](#) ([Time](#)) const
- [Rate](#) [compoundForwardImpl](#) ([Time](#), [Integer](#)) const

7.216.2 Constructor & Destructor Documentation

7.216.2.1 [ForwardRateStructure](#) (const [Date](#) & *today'sDate*, const [Date](#) & *referenceDate*)

Deprecated

use the constructor without today's date; set the evaluation date through [Settings::instance\(\)](#).

7.216.3 Member Function Documentation

7.216.3.1 [DiscountFactor](#) discountImpl ([Time](#)) const [protected, virtual]

Returns the discount factor for the given date calculating it from the instantaneous forward rate.

Implements [YieldTermStructure](#).

Reimplemented in [CompoundForward](#).

7.216.3.2 [Rate](#) zeroYieldImpl ([Time](#)) const [protected, virtual]

Returns the zero yield rate for the given date calculating it from the instantaneous forward rate.

Warning:

This is just a default, highly inefficient and possibly wildly inaccurate implementation. Derived classes should implement their own zeroYield method.

Implements [YieldTermStructure](#).

Reimplemented in [CompoundForward](#), and [ForwardSpreadedTermStructure](#).

7.216.3.3 [Rate](#) compoundForwardImpl ([Time](#), [Integer](#)) const [protected, virtual]

Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

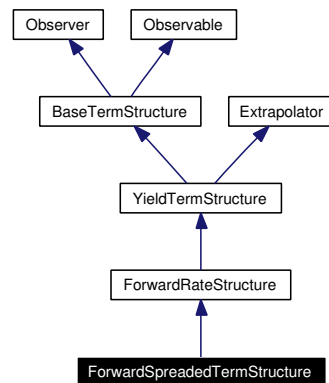
Implements [YieldTermStructure](#).

Reimplemented in [CompoundForward](#).

7.217 ForwardSpreadedTermStructure Class Reference

```
#include <ql/TermStructures/forwardspreadedtermstructure.hpp>
```

Inheritance diagram for ForwardSpreadedTermStructure:



7.217.1 Detailed Description

Term structure with added spread on the instantaneous forward rate.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Tests

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Public Member Functions

- **ForwardSpreadedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Handle](#)< [Quote](#) > &spread)

YieldTermStructure interface

- [DayCounter](#) **dayCounter** () const
the day counter used for date/time conversion
- [Calendar](#) **calendar** () const
the calendar used for reference date calculation
- const [Date](#) & **todaysDate** () const
today's date
- const [Date](#) & **referenceDate** () const
the reference date, i.e., the date at which discount = 1

- [Date maxDate \(\)](#) const
the latest date for which the curve can return rates
- [Time maxTime \(\)](#) const
the latest time for which the curve can return rates

Protected Member Functions

- [Rate forwardImpl \(Time\)](#) const
returns the spreaded forward rate
- [Rate zeroYieldImpl \(Time\)](#) const
returns the spreaded zero yield rate

7.217.2 Member Function Documentation

7.217.2.1 [const Date & todaysDate \(\)](#) const [virtual]

today's date

Deprecated

use [Settings::instance\(\).evaluationDate\(\)](#).

Reimplemented from [BaseTermStructure](#).

7.217.2.2 [Rate zeroYieldImpl \(Time\)](#) const [protected, virtual]

returns the spreaded zero yield rate

Warning:

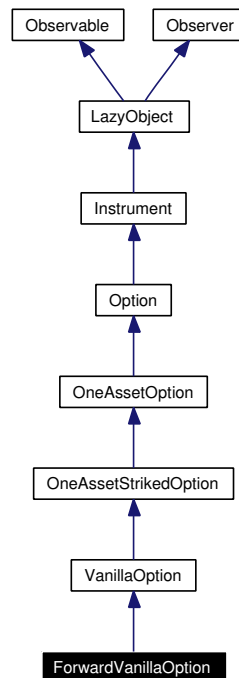
This method must disappear should the spread become a curve

Reimplemented from [ForwardRateStructure](#).

7.218 ForwardVanillaOption Class Reference

```
#include <ql/Instruments/forwardvanillaoption.hpp>
```

Inheritance diagram for ForwardVanillaOption:



7.218.1 Detailed Description

Forward version of a vanilla option.

Public Types

- typedef [ForwardOptionArguments](#)< VanillaOption::arguments > **arguments**
- typedef VanillaOption::results **results**
- typedef [ForwardEngine](#)< VanillaOption::arguments, VanillaOption::results > **engine**

Public Member Functions

- **ForwardVanillaOption** ([Real](#) moneyness, [Date](#) resetDate, const boost::shared_ptr< [BlackScholesProcess](#) > &stochProc, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine)
- void [setupArguments](#) ([Arguments](#) *) const

Protected Member Functions

- void [performCalculations](#) () const

7.218.2 Member Function Documentation

7.218.2.1 void setupArguments ([Arguments](#) *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetStrikedOption](#).

7.218.2.2 void performCalculations () const [protected, virtual]

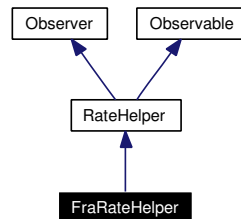
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [OneAssetStrikedOption](#).

7.219 FraRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for FraRateHelper:



7.219.1 Detailed Description

Forward rate agreement.

Warning:

This class assumes that the reference date does not change between calls of `setTermStructure()`.

Todo

convexity adjustment should be implemented.

Public Member Functions

- **FraRateHelper** (const [Handle](#)< [Quote](#) > &rate, [Integer](#) monthsToStart, [Integer](#) monthsToEnd, [Integer](#) settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **FraRateHelper** ([Rate](#) rate, [Integer](#) monthsToStart, [Integer](#) monthsToEnd, [Integer](#) settlementDays, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- [Real](#) **impliedQuote** () const
- [DiscountFactor](#) **discountGuess** () const
- void **setTermStructure** ([YieldTermStructure](#) *)
sets the term structure to be used for pricing
- [Date](#) **latestDate** () const
latest relevant date

7.219.2 Member Function Documentation

7.219.2.1 void setTermStructure ([YieldTermStructure](#) *) [virtual]

sets the term structure to be used for pricing

Warning:

Being a pointer and not a shared_ptr, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that

the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

7.219.2.2 [Date](#) latestDate () const [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

7.220 FrequencyFormatter Class Reference

```
#include <ql/date.hpp>
```

7.220.1 Detailed Description

Formats frequency for output.

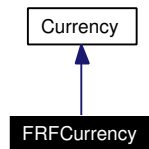
Static Public Member Functions

- `std::string toString` ([Frequency](#) f)

7.221 FRFCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for FRFCurrency:



7.221.1 Detailed Description

French franc.

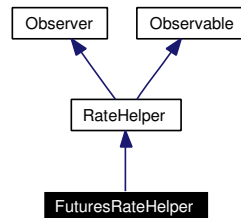
The ISO three-letter code is FRF; the numeric code is 250. It is divided in 100 centimes.

ingroup currencies

7.222 FuturesRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for FuturesRateHelper:



7.222.1 Detailed Description

Interest-rate futures.

Warning:

This class assumes that the reference date does not change between calls of [setTermStructure\(\)](#).

Public Member Functions

- **FuturesRateHelper** (const [Handle](#)< [Quote](#) > &price, const [Date](#) &immDate, [Integer](#) nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **FuturesRateHelper** (const [Handle](#)< [Quote](#) > &price, const [Date](#) &immDate, const [Date](#) &matDate, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **FuturesRateHelper** ([Real](#) price, const [Date](#) &immDate, [Integer](#) nMonths, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention, const [DayCounter](#) &dayCounter)
- **Real impliedQuote** () const
- **DiscountFactor discountGuess** () const
- **Date latestDate** () const

latest relevant date

7.222.2 Member Function Documentation

7.222.2.1 [Date](#) latestDate () const [virtual]

latest relevant date

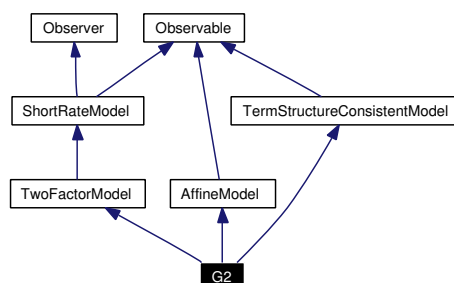
The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

7.223 G2 Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Inheritance diagram for G2:



7.223.1 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 [Handle< YieldTermStructure >](#) &termStructure, [Real](#) a=0.1, [Real](#) sigma=0.01, [Real](#) b=0.1, [Real](#) eta=0.01, [Real](#) rho=-0.75)
- [boost::shared_ptr< ShortRateDynamics >](#) **dynamics** () const

Returns the short-rate dynamics.

- [Real](#) **discountBondOption** ([Option::Type](#) type, [Real](#) strike, [Time](#) maturity, [Time](#) bond-Maturity) const
- [Real](#) **swaption** (const [Swaption::arguments](#) &arguments, [Real](#) range, [Size](#) intervals) const
- [DiscountFactor](#) **discount** ([Time](#) t) const

Implied discount curve.

Protected Member Functions

- void **generateArguments** ()
- [Real](#) **A** ([Time](#) t, [Time](#) T) const
- [Real](#) **B** ([Real](#) x, [Time](#) t) const

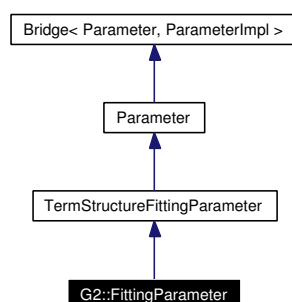
Friends

- class `SwaptionPricingFunction`

7.224 G2::FittingParameter Class Reference

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Inheritance diagram for G2::FittingParameter:



7.224.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) + \frac{1}{2} \left(\frac{\sigma(1 - e^{-at})}{a} \right)^2 + \frac{1}{2} \left(\frac{\eta(1 - e^{-bt})}{b} \right)^2 + \rho \frac{\sigma(1 - e^{-at})}{a} \frac{\eta(1 - e^{-bt})}{b},$$

where $f(t)$ is the instantaneous forward rate at t .

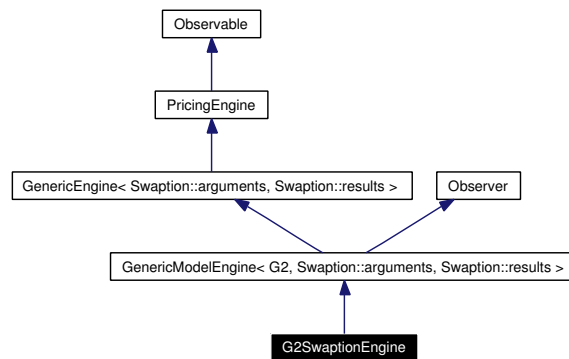
Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a, [Real](#) sigma, [Real](#) b, [Real](#) eta, [Real](#) rho)

7.225 G2SwaptionEngine Class Reference

#include <ql/PricingEngines/Swaption/g2swaptionengine.hpp>

Inheritance diagram for G2SwaptionEngine:



7.225.1 Detailed Description

Swaption priced by means of the Black formula

Public Member Functions

- **G2SwaptionEngine** (const boost::shared_ptr< [G2](#) > &mod, [Real](#) range, [Size](#) intervals)
- void **calculate** () const

7.226 GammaFunction Class Reference

```
#include <ql/Math/gammadistribution.hpp>
```

7.226.1 Detailed Description

Gamma function class.

This is a function defined by

$$\Gamma(z) = \int_0^{\infty} t^{z-1} e^{-t} dt$$

The implementation of the algorithm was inspired by "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery, chapter 6

Tests

the correctness of the returned value is tested by checking it against known good results.

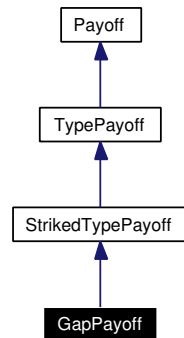
Public Member Functions

- [Real](#) logValue ([Real](#) x) const

7.227 GapPayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for GapPayoff:



7.227.1 Detailed Description

Binary gap payoff.

Public Member Functions

- **GapPayoff** (Option::Type type, [Real](#) strike, [Real](#) strikePayoff)
- [Real](#) **operator()** ([Real](#) price) const
- [Real](#) **strikePayoff** () const

7.228 GaussianStatistics Class Template Reference

```
#include <ql/Math/gaussianstatistics.hpp>
```

7.228.1 Detailed Description

template<class Stat> class QuantLib::GaussianStatistics< Stat >

Statistics tool for gaussian-assumption risk measures.

It can calculate gaussian assumption risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the mean and variance provided by the template class

Public Member Functions

- **GaussianStatistics** (const Stat &s)

Gaussian risk measures

- [Real gaussianDownsideVariance](#) () const
- [Real gaussianDownsideDeviation](#) () const
- [Real gaussianRegret](#) (Real target) const
- [Real gaussianPercentile](#) (Real percentile) const
- [Real gaussianTopPercentile](#) (Real percentile) const
- [Real gaussianPotentialUpside](#) (Real percentile) const
gaussian-assumption Potential-Upside at a given percentile
- [Real gaussianValueAtRisk](#) (Real percentile) const
gaussian-assumption Value-At-Risk at a given percentile
- [Real gaussianExpectedShortfall](#) (Real percentile) const
gaussian-assumption Expected Shortfall at a given percentile
- [Real gaussianShortfall](#) (Real target) const
gaussian-assumption Shortfall (observations below target)
- [Real gaussianAverageShortfall](#) (Real target) const
gaussian-assumption Average Shortfall (averaged shortfallness)

7.228.2 Member Function Documentation

7.228.2.1 [Real gaussianDownsideVariance](#) () const

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^N \theta \times x_i^2}{\sum_{i=1}^N w_i}$$

, where $\theta = 0$ if $x > 0$ and $\theta = 1$ if $x < 0$

7.228.2.2 Real gaussianDownsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.228.2.3 Real gaussianRegret (Real target) const

returns the variance of observations below target

$$\frac{\sum w_i (\min(0, x_i - \text{target}))^2}{\sum w_i}.$$

See Dembo, Freeman "The Rules Of Risk", Wiley (2001)

7.228.2.4 Real gaussianPercentile (Real percentile) const

gaussian-assumption y-th percentile, defined as the value x such that

$$y = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x \exp(-u^2/2) du$$

7.228.2.5 Real gaussianTopPercentile (Real percentile) const

Precondition:

percentile must be in range (0%-100%) extremes excluded

7.228.2.6 Real gaussianPotentialUpside (Real percentile) const

gaussian-assumption Potential-Upside at a given percentile

Precondition:

percentile must be in range [90%-100%)

7.228.2.7 Real gaussianValueAtRisk (Real percentile) const

gaussian-assumption Value-At-Risk at a given percentile

Precondition:

percentile must be in range [90%-100%)

7.228.2.8 Real gaussianExpectedShortfall (Real percentile) const

gaussian-assumption Expected Shortfall at a given percentile

Assuming a gaussian distribution it returns the expected loss in case that the loss exceeded a VaR threshold,

$$E[x \mid x < \text{VaR}(p)],$$

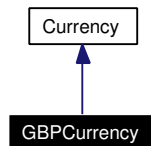
that is the average of observations below the given percentile p . Also known as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

7.229 GBPCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for GBPCurrency:



7.229.1 Detailed Description

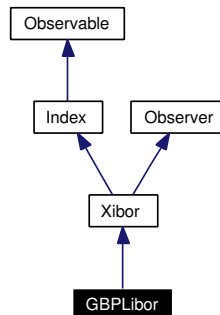
British pound sterling.

The ISO three-letter code is GBP; the numeric code is 826. It is divided into 100 pence.

7.230 GBPLibor Class Reference

```
#include <ql/Indexes/gbplibor.hpp>
```

Inheritance diagram for GBPLibor:



7.230.1 Detailed Description

GBP Libor index

Public Member Functions

- **GBPLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual365Fixed](#)())

7.231 GeneralStatistics Class Reference

```
#include <ql/Math/generalstatistics.hpp>
```

7.231.1 Detailed Description

Statistics tool.

This class accumulates a set of data and returns their statistics (e.g: mean, variance, skewness, kurtosis, error estimation, percentile, etc.) based on the empirical distribution (no gaussian assumption)

It doesn't suffer the numerical instability problem of [IncrementalStatistics](#). The downside is that it stores all samples, thus increasing the memory requirements.

Public Member Functions

Inspectors

- [Size samples](#) () const
number of samples collected
- const std::vector< std::pair< [Real](#), [Real](#) > > & [data](#) () const
collected data
- [Real weightSum](#) () const
sum of data weights
- [Real mean](#) () const
- [Real variance](#) () const
- [Real standardDeviation](#) () const
- [Real errorEstimate](#) () const
- [Real skewness](#) () const
- [Real kurtosis](#) () const
- [Real min](#) () const
- [Real max](#) () const
- template<class Func, class Predicate> std::pair< [Real](#), [Size](#) > [expectationValue](#) (const Func &f, const Predicate &inRange) const
- [Real percentile](#) ([Real](#) y) const
- [Real topPercentile](#) ([Real](#) y) const

Modifiers

- void [add](#) ([Real](#) value, [Real](#) weight=1.0)
adds a datum to the set, possibly with a weight
- template<class DataIterator> void [addSequence](#) (DataIterator begin, DataIterator end)
adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator> void [addSequence](#) (DataIterator begin, DataIterator end, WeightIterator wbegin)
adds a sequence of data to the set, each with its weight
- void [reset](#) ()

resets the data to a null set

- void `sort ()` const
sort the data set in increasing order

7.231.2 Member Function Documentation

7.231.2.1 `Real mean ()` const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

7.231.2.2 `Real variance ()` const

returns the variance, defined as

$$\sigma^2 = \frac{N}{N-1} \langle (x - \langle x \rangle)^2 \rangle.$$

7.231.2.3 `Real standardDeviation ()` const

returns the standard deviation σ , defined as the square root of the variance.

7.231.2.4 `Real errorEstimate ()` const

returns the error estimate on the mean value, defined as $\epsilon = \sigma / \sqrt{N}$.

7.231.2.5 `Real skewness ()` const

returns the skewness, defined as

$$\frac{N^2}{(N-1)(N-2)} \frac{\langle (x - \langle x \rangle)^3 \rangle}{\sigma^3}.$$

The above evaluates to 0 for a Gaussian distribution.

7.231.2.6 `Real kurtosis ()` const

returns the excess kurtosis, defined as

$$\frac{N^2(N+1)}{(N-1)(N-2)(N-3)} \frac{\langle (x - \langle x \rangle)^4 \rangle}{\sigma^4} - \frac{3(N-1)^2}{(N-2)(N-3)}.$$

The above evaluates to 0 for a Gaussian distribution.

7.231.2.7 Real min () const

returns the minimum sample value

7.231.2.8 Real max () const

returns the maximum sample value

7.231.2.9 std::pair<Real,Size> expectationValue (const Func &f, const Predicate &inRange) const

Expectation value of a function f on a given range \mathcal{R} , i.e.,

$$E[f | \mathcal{R}] = \frac{\sum_{x_i \in \mathcal{R}} f(x_i) w_i}{\sum_{x_i \in \mathcal{R}} w_i}.$$

The range is passed as a boolean function returning `true` if the argument belongs to the range or `false` otherwise.

The function returns a pair made of the result and the number of observations in the given range.

7.231.2.10 Real percentile (Real y) const

y -th percentile, defined as the value \bar{x} such that

$$y = \frac{\sum_{x_i < \bar{x}} w_i}{\sum_i w_i}$$

Precondition:

y must be in the range $(0 - 1]$.

7.231.2.11 Real topPercentile (Real y) const

y -th top percentile, defined as the value \bar{x} such that

$$y = \frac{\sum_{x_i > \bar{x}} w_i}{\sum_i w_i}$$

Precondition:

y must be in the range $(0 - 1]$.

7.231.2.12 void add (Real value, Real weight = 1.0)

adds a datum to the set, possibly with a weight

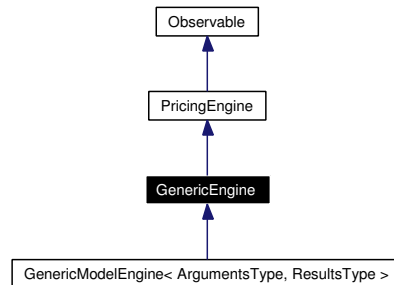
Precondition:

weights must be positive or null

7.232 GenericEngine Class Template Reference

```
#include <ql/pricingengine.hpp>
```

Inheritance diagram for GenericEngine:



7.232.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::GenericEngine<
ArgumentsType, ResultsType >
```

template base class for option pricing engines

Derived engines only need to implement the `calculate()` method.

Public Member Functions

- `Arguments * arguments () const`
- `const Results * results () const`
- `void reset () const`

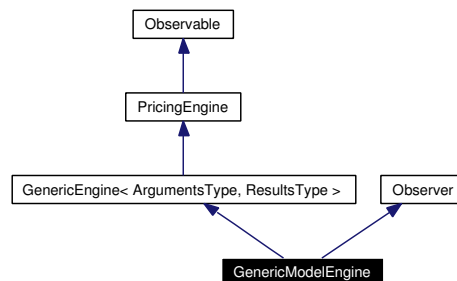
Protected Attributes

- `ArgumentsType arguments_`
- `ResultsType results_`

7.233 GenericModelEngine Class Template Reference

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Inheritance diagram for GenericModelEngine:



7.233.1 Detailed Description

```
template<class ModelType, class ArgumentsType, class ResultsType> class QuantLib::GenericModelEngine< ModelType, ArgumentsType, ResultsType >
```

Base class for some pricing engine on a particular model.

Derived engines only need to implement the `calculate()` method

Public Member Functions

- **GenericModelEngine** (const boost::shared_ptr< ModelType > &model)
- void **setModel** (const boost::shared_ptr< ModelType > &model)
- virtual void **update** ()

Protected Attributes

- boost::shared_ptr< ModelType > **model_**

7.233.2 Member Function Documentation

7.233.2.1 virtual void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

Reimplemented in [LatticeShortRateModelEngine](#), [LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >](#), and [LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >](#).

7.234 GenericRiskStatistics Class Template Reference

```
#include <ql/Math/riskstatistics.hpp>
```

7.234.1 Detailed Description

template<class S> class QuantLib::GenericRiskStatistics< S >

empirical-distribution risk measures

This class wraps a somewhat generic statistic tool and adds a number of risk measures (e.g.: value-at-risk, expected shortfall, etc.) based on the data distribution as reported by the underlying tool.

Todo

add historical annualized volatility

Public Member Functions

- [Real semiVariance](#) () const
- [Real semiDeviation](#) () const
- [Real downsideVariance](#) () const
- [Real downsideDeviation](#) () const
- [Real regret](#) (Real target) const
- [Real potentialUpside](#) (Real percentile) const
potential upside (the reciprocal of VAR) at a given percentile
- [Real valueAtRisk](#) (Real percentile) const
value-at-risk at a given percentile
- [Real expectedShortfall](#) (Real percentile) const
expected shortfall at a given percentile
- [Real shortfall](#) (Real target) const
- [Real averageShortfall](#) (Real target) const

7.234.2 Member Function Documentation

7.234.2.1 [Real semiVariance](#) () const

returns the variance of observations below the mean,

$$\frac{N}{N-1} \mathbb{E} \left[(x - \langle x \rangle)^2 \mid x < \langle x \rangle \right].$$

See Markowitz (1959).

7.234.2.2 [Real semiDeviation](#) () const

returns the semi deviation, defined as the square root of the semi variance.

7.234.2.3 Real downsideVariance () const

returns the variance of observations below 0.0,

$$\frac{N}{N-1} E[x^2 \mid x < 0].$$

7.234.2.4 Real downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.234.2.5 Real regret (Real target) const

returns the variance of observations below target,

$$\frac{N}{N-1} E[(x - t)^2 \mid x < t].$$

See Dembo and Freeman, "The Rules Of Risk", Wiley (2001).

7.234.2.6 Real potentialUpside (Real centile) const

potential upside (the reciprocal of VAR) at a given percentile

Precondition:

percentile must be in range [90%-100%)

7.234.2.7 Real valueAtRisk (Real centile) const

value-at-risk at a given percentile

Precondition:

percentile must be in range [90%-100%)

7.234.2.8 Real expectedShortfall (Real percentile) const

expected shortfall at a given percentile

returns the expected loss in case that the loss exceeded a VaR threshold,

$$E[x \mid x < \text{VaR}(p)],$$

that is the average of observations below the given percentile p . Also know as conditional value-at-risk.

See Artzner, Delbaen, Eber and Heath, "Coherent measures of risk", Mathematical Finance 9 (1999)

7.234.2.9 Real shortfall (Real target) const

probability of missing the given target, defined as

$$E[\Theta \mid (-\infty, \infty)]$$

where

$$\Theta(x) = \begin{cases} 1 & x < t \\ 0 & x \geq t \end{cases}$$

7.234.2.10 Real averageShortfall (Real target) const

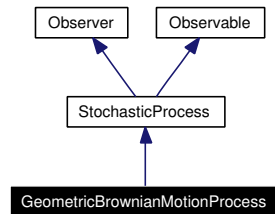
averaged shortfallness, defined as

$$E[t - x \mid x < t]$$

7.235 GeometricBrownianMotionProcess Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for GeometricBrownianMotionProcess:



7.235.1 Detailed Description

Geometric brownian motion process.

This class describes the stochastic process governed by

$$dS(t, S) = \mu S dt + \sigma S dW_t.$$

Public Member Functions

- **GeometricBrownianMotionProcess** (double initialValue, double mue, double sigma)
- **Real x0** () const
returns the initial value of the state variable
- **Real drift** (Time t, Real x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- **Real diffusion** (Time t, Real x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

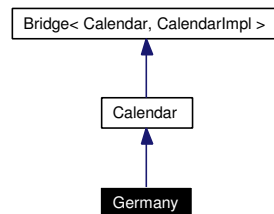
Protected Attributes

- double **initialValue_**
- double **mue_**
- double **sigma_**

7.236 Germany Class Reference

```
#include <ql/Calendars/germany.hpp>
```

Inheritance diagram for Germany:



7.236.1 Detailed Description

German calendars.

Public holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Ascension Thursday
- Whit Monday
- Corpus Christi
- Labour Day, May 1st
- National Day, October 3rd
- Christmas Eve, December 24th
- Christmas, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31st

Holidays for the Frankfurt [Stock](http://deutsche-boerse.com/) exchange (data from <http://deutsche-boerse.com/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday

- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

Holidays for the Xetra exchange (data from <http://deutsche-boerse.com/>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

Holidays for the Eurex exchange (data from <http://www.eurexchange.com/index.html>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday
- Labour Day, May 1st
- Christmas' Eve, December 24th
- Christmas, December 25th
- Christmas Holiday, December 26th
- New Year's Eve, December 31st

Tests

the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum [Market](#) { [Settlement](#), [FrankfurtStockExchange](#), [Xetra](#), [Eurex](#) }
German calendars.

Public Member Functions

- [Germany](#) ([Market](#) market=[FrankfurtStockExchange](#))

7.236.2 Member Enumeration Documentation

7.236.2.1 enum [Market](#)

German calendars.

Enumeration values:

Settlement generic settlement calendar

FrankfurtStockExchange Frankfurt stock-exchange.

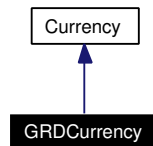
Xetra Xetra.

Eurex Eurex.

7.237 GRDCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for GRDCurrency:



7.237.1 Detailed Description

Greek drachma.

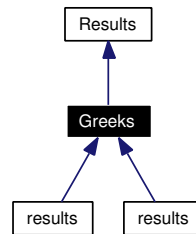
The ISO three-letter code is GRD; the numeric code is 300. It is divided in 100 lepta.

ingroup currencies

7.238 Greeks Class Reference

```
#include <ql/option.hpp>
```

Inheritance diagram for Greeks:



7.238.1 Detailed Description

additional option results

Public Member Functions

- `void reset ()`

Public Attributes

- [Real](#) `delta`
- [Real](#) `gamma`
- [Real](#) `theta`
- [Real](#) `vega`
- [Real](#) `rho`
- [Real](#) `dividendRho`

7.239 HaltonRsg Class Reference

```
#include <ql/RandomNumbers/haltonrsg.hpp>
```

7.239.1 Detailed Description

Halton low-discrepancy sequence generator.

Halton algorithm for low-discrepancy sequence. For more details see chapter 8, paragraph 2 of "Monte Carlo Methods in Finance", by Peter Jäckel

Tests

- a) the correctness of the returned values is tested by reproducing known good values.
- b) the correctness of the returned values is tested by checking their discrepancy against known good values.

Public Types

- typedef [Sample](#)< [Array](#) > **sample_type**

Public Member Functions

- **HaltonRsg** ([Size](#) dimensionality, unsigned long seed=0, bool randomStart=true, bool randomShift=false)
- const [sample_type](#) & **nextSequence** () const
- const [sample_type](#) & **lastSequence** () const
- [Size](#) **dimension** () const

7.240 Handle Class Template Reference

```
#include <ql/relinkablehandle.hpp>
```

7.240.1 Detailed Description

template<class Type> class QuantLib::Handle< Type >

Globally accessible relinkable pointer.

An instance of this class can be relinked to another shared pointer: such change will be propagated to all the copies of the instance.

Precondition:

Class "Type" must inherit from [Observable](#)

Public Member Functions

- [Handle](#) (const boost::shared_ptr< Type > &h=boost::shared_ptr< Type >(), bool registerAsObserver=true)
- void [linkTo](#) (const boost::shared_ptr< Type > &, bool registerAsObserver=true)
- const boost::shared_ptr< Type > & [currentLink](#) () const
dereferencing
- const boost::shared_ptr< Type > & **operator** → () const
- bool [empty](#) () const
Checks if the contained shared pointer points to anything.
- bool [isNull](#) () const
Checks if the contained shared pointer points to anything.

7.240.2 Constructor & Destructor Documentation

7.240.2.1 [Handle](#) (const boost::shared_ptr< Type > &h = boost::shared_ptr< Type >(), bool registerAsObserver = true) [explicit]

Warning:

see the documentation of the [Link](#) class for issues relatives to registerAsObserver.

7.240.3 Member Function Documentation

7.240.3.1 void [linkTo](#) (const boost::shared_ptr< Type > &, bool registerAsObserver = true)

Warning:

see the documentation of the [Link](#) class for issues relatives to registerAsObserver.

7.240.3.2 `bool isNull () const`

Checks if the contained shared pointer points to anything.

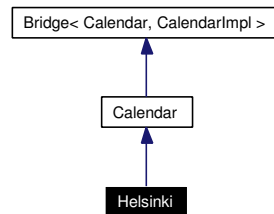
Deprecated

use `empty()` instead

7.241 Helsinki Class Reference

```
#include <ql/Calendars/helsinki.hpp>
```

Inheritance diagram for Helsinki:



7.241.1 Detailed Description

Helsinki calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Good Friday
- Easter Monday
- Ascension Thursday
- Labour Day, May 1st
- Midsummer Eve (Friday between June 18-24)
- Independence Day, December 6th
- Christmas Eve, December 24th
- Christmas, December 25th
- Boxing Day, December 26th

7.242 History Class Reference

```
#include <ql/history.hpp>
```

7.242.1 Detailed Description

Container for historical data.

This class acts as a generic repository for a set of historical data. Single data can be accessed through their date, while sets of consecutive data can be accessed through iterators.

A history can contain null data, which can either be returned or skipped according to the chosen iterator type.

Example: [uses of history iterators](#)

Public Types

- `typedef boost::filter_iterator< DataValidator, const_iterator > const_valid_iterator`
bidirectional iterator on non-null history entries
- `typedef std::vector< Real >::const_iterator const_data_iterator`
random access iterator on historical data
- `typedef boost::filter_iterator< DataValidator, const_data_iterator > const_valid_data_iterator`
bidirectional iterator on non-null historical data

Public Member Functions

- [History](#) ()
- `template<class Iterator> History (const Date &firstDate, const Date &lastDate, Iterator begin, Iterator end)`
- `History (const Date &firstDate, const std::vector< Real > &values)`
- `History (const Date &firstDate, const Date &lastDate, const std::vector< Real > &values)`
- `History (const std::vector< Date > &dates, const std::vector< Real > &values)`

Inspectors

- `const Date & firstDate () const`
returns the first date for which a historical datum exists
- `const Date & lastDate () const`
returns the last date for which a historical datum exists
- `Size size () const`
returns the number of historical data including null ones

Historical data access

- **Real operator[]** (const **Date** &) const
returns the (possibly null) datum corresponding to the given date

Iterator access

Four different types of iterators are provided, namely, `const_iterator`, `const_valid_iterator`, `const_data_iterator`, and `const_valid_data_iterator`.

`const_iterator` and `const_valid_iterator` point to an `Entry` structure, the difference being that the latter only iterates over valid entries - i.e., entries whose data are not null. The same difference exists between `const_data_iterator` and `const_valid_data_iterator` which point directly to historical values without reference to the date they are associated to.

- **const_iterator begin** () const
- **const_iterator end** () const
- **const_iterator iterator** (const **Date** &d) const
- **const_valid_iterator vbegin** () const
- **const_valid_iterator vend** () const
- **const_valid_iterator valid_iterator** (const **Date** &d) const
- **const_data_iterator dbegin** () const
- **const_data_iterator dend** () const
- **const_data_iterator data_iterator** (const **Date** &d) const
- **const_valid_data_iterator vdbegin** () const
- **const_valid_data_iterator vdend** () const
- **const_valid_data_iterator valid_data_iterator** (const **Date** &d) const

7.242.2 Constructor & Destructor Documentation

7.242.2.1 **History** ()

Default constructor

7.242.2.2 **History** (const **Date** & *firstDate*, const **Date** & *lastDate*, Iterator *begin*, Iterator *end*)

This constructor initializes the history with the given set of values, corresponding to the date range between *firstDate* and *lastDate* included.

Precondition:

begin-end must equal the number of days from *firstDate* to *lastDate* included.

7.242.2.3 **History** (const **Date** & *firstDate*, const **Date** & *lastDate*, const std::vector< **Real** > & *values*)

This constructor initializes the history with the given set of values, corresponding to the date range between *firstDate* and *lastDate* included.

Precondition:

The size of *values* must equal the number of days from *firstDate* to *lastDate* included.

7.242.2.4 History (const std::vector< Date > & *dates*, const std::vector< Real > & *values*)

This constructor initializes the history with the given set of values, corresponding each to the element with the same index in the given set of dates. The whole date range between *dates*[0] and *dates*[N-1] will be automatically filled by inserting null values where a date is missing from the given set.

Precondition:

dates must be sorted.

There can be no pairs (*dates*[i],*values*[i]) and (*dates*[j],*values*[j]) such that *dates*[i] == *dates*[j] && *values*[i] != *values*[j]. Pairs with *dates*[i] == *dates*[j] && *values*[i] == *values*[j] are allowed; the duplicated entries will be discarded.

The size of *values* must equal the number of days from *firstDate* to *lastDate* included.

7.243 History::const_iterator Class Reference

```
#include <ql/history.hpp>
```

7.243.1 Detailed Description

random access iterator on history entries

Public Member Functions

- const [Entry](#) & **dereference** () const
- bool **equal** (const [const_iterator](#) &i) const
- void **increment** ()
- void **decrement** ()
- void **advance** ([BigInteger](#) n)
- [BigInteger](#) **distance_to** (const [const_iterator](#) &i) const

Friends

- class **History**

7.244 History::Entry Class Reference

```
#include <ql/history.hpp>
```

7.244.1 Detailed Description

single datum in history

Public Member Functions

- const [Date](#) & **date** () const
- [Real](#) **value** () const

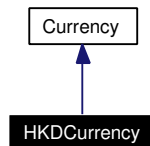
Friends

- class **const_iterator**

7.245 HKDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for HKDCurrency:



7.245.1 Detailed Description

Honk Kong dollar.

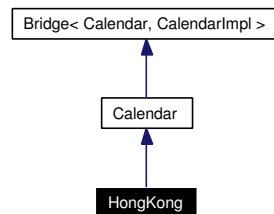
The ISO three-letter code is HKD; the numeric code is 344. It is divided in 100 cents.

ingroup currencies

7.246 HongKong Class Reference

```
#include <ql/Calendars/hongkong.hpp>
```

Inheritance diagram for HongKong:



7.246.1 Detailed Description

Hong Kong calendar.

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Ching Ming Festival, April 5th
- Good Friday
- Easter Monday
- Labor Day, May 1st
- SAR Establishment Day, July 1st
- National Day, October 1st
- Christmas, December 25th
- Boxing Day, December 26th
- Christmas Holiday, December 27th

Other holidays for which no rule is given (data available for 2004/2005 only:)

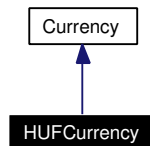
- Lunar New Year
- Chinese New Year
- Buddha's birthday
- Tuen NG Festival
- Mid-autumn fest
- Chung Yeung fest

Data from <http://www.hkex.com.hk>

7.247 HUFCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for HUFCurrency:



7.247.1 Detailed Description

Hungarian forint.

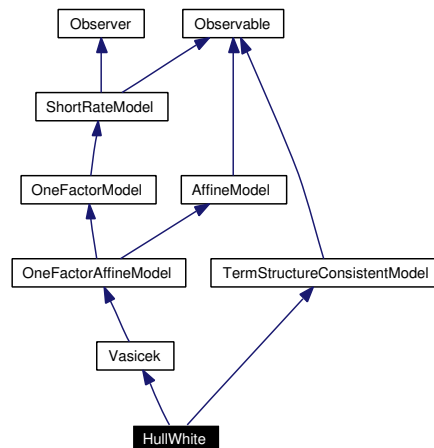
The ISO three-letter code is HUF; the numeric code is 348. It has no subdivisions.

ingroup currencies

7.248 HullWhite Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/hullwhite.hpp>
```

Inheritance diagram for HullWhite:



7.248.1 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.

Bug

When the term structure is relinked, the `r0` parameter of the underlying [Vasicek](#) model is not updated.

Public Member Functions

- **HullWhite** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a=0.1, [Real](#) sigma=0.01)
- `boost::shared_ptr< Lattice > tree` (const [TimeGrid](#) &grid) const
Return by default a trinomial recombining tree.
- `boost::shared_ptr< ShortRateDynamics > dynamics` () const
returns the short-rate dynamics
- **[Real](#) discountBondOption** ([Option::Type](#) type, [Real](#) strike, [Time](#) maturity, [Time](#) bond-Maturity) const

Protected Member Functions

- void **generateArguments** ()
- **[Real](#) A** ([Time](#) t, [Time](#) T) const

7.249 HullWhite::Dynamics Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/hullwhite.hpp>
```

7.249.1 Detailed Description

Short-rate dynamics in the Hull-White model.

The short-rate is here

$$r_t = \varphi(t) + x_t$$

where $\varphi(t)$ is the deterministic time-dependent parameter used for term-structure fitting and x_t is the state variable following an Ornstein-Uhlenbeck process.

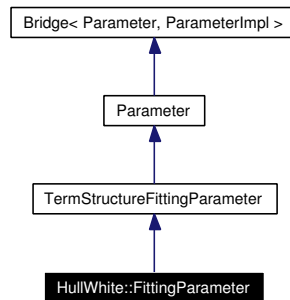
Public Member Functions

- **Dynamics** (const [Parameter](#) &fitting, [Real](#) a, [Real](#) sigma)
- **Real variable** ([Time](#) t, [Rate](#) r) const
- **Real shortRate** ([Time](#) t, [Real](#) x) const

7.250 HullWhite::FittingParameter Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/hullwhite.hpp>
```

Inheritance diagram for HullWhite::FittingParameter:



7.250.1 Detailed Description

Analytical term-structure fitting parameter $\varphi(t)$.

$\varphi(t)$ is analytically defined by

$$\varphi(t) = f(t) + \frac{1}{2} \left[\frac{\sigma(1 - e^{-at})}{a} \right]^2,$$

where $f(t)$ is the instantaneous forward rate at t .

Public Member Functions

- **FittingParameter** (const [Handle](#)< [YieldTermStructure](#) > &termStructure, [Real](#) a, [Real](#) sigma)

7.251 ICGaussianRng Class Template Reference

```
#include <ql/RandomNumbers/inversecumgaussianrng.hpp>
```

7.251.1 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();  
Real I::operator() const;
```

Deprecated

use [InverseCumulativeRng](#) instead

Public Types

- typedef [Sample](#)< [Real](#) > **sample_type**
- typedef RNG **urng_type**

Public Member Functions

- **ICGaussianRng** (const RNG &uniformGenerator)
- [sample_type next](#) () const
returns a sample from a Gaussian distribution

7.252 ICGaussianRsg Class Template Reference

```
#include <ql/RandomNumbers/inversecumgaussianrsg.hpp>
```

7.252.1 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::nextSequence() const;
Size USG::dimension() const;
```

The inverse cumulative normal distribution is supplied by I.

Class I must implement the following interface:

```
I::I();
Real I::operator() const;
```

Deprecated

use [InverseCumulativeRsg](#) instead

Public Types

- typedef [Sample< Array >](#) **sample_type**

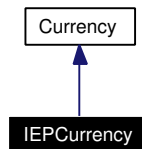
Public Member Functions

- **ICGaussianRsg** (const USG &uniformSequenceGenerator)
- **ICGaussianRsg** (const USG &uniformSequenceGenerator, const I &inverseCumulative)
- const [sample_type](#) & [nextSequence](#) () const
returns next sample from the Gaussian distribution
- const [sample_type](#) & [lastSequence](#) () const
- [Size](#) [dimension](#) () const

7.253 IEPCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for IEPCurrency:



7.253.1 Detailed Description

Irish punt.

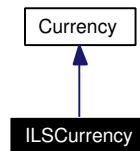
The ISO three-letter code is IEP; the numeric code is 372. It is divided in 100 pence.

ingroup currencies

7.254 ILSCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for ILSCurrency:



7.254.1 Detailed Description

Israeli shekel.

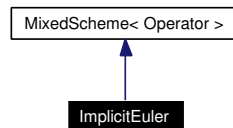
The ISO three-letter code is ILS; the numeric code is 376. It is divided in 100 agorot.

ingroup currencies

7.255 ImplicitEuler Class Template Reference

```
#include <ql/FiniteDifferences/impliciteuler.hpp>
```

Inheritance diagram for ImplicitEuler:



7.255.1 Detailed Description

```
template<class Operator> class QuantLib::ImplicitEuler< Operator >
```

Backward Euler scheme for finite difference methods.

See sect. [Finite-differences framework](#) for details on the method.

In this implementation, the passed operator must be derived from either `TimeConstantOperator` or `TimeDependentOperator`. Also, it must implement at least the following interface:

```
typedef ... 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*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

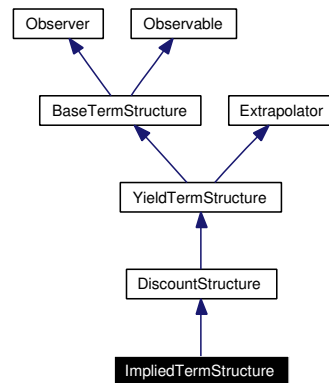
Friends

- class `FiniteDifferenceModel<ImplicitEuler<Operator> >`

7.256 ImpliedTermStructure Class Reference

```
#include <ql/TermStructures/impliedtermstructure.hpp>
```

Inheritance diagram for ImpliedTermStructure:



7.256.1 Detailed Description

Implied term structure at a given date in the future.

The given date will be the implied reference date.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Tests

- a) the correctness of the returned values is tested by checking them against numerical calculations.
- b) observability against changes in the underlying term structure is checked.

Public Member Functions

- **ImpliedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Date](#) &newTodayDate, const [Date](#) &newReferenceDate)
- **ImpliedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Date](#) &referenceDate)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- [Date](#) [maxDate](#) () const
the latest date for which the curve can return rates

Protected Member Functions

- [DiscountFactor](#) [discountImpl](#) ([Time](#)) const
returns the discount factor as seen from the evaluation date

7.256.2 Constructor & Destructor Documentation

- 7.256.2.1 [ImpliedTermStructure](#) (const [Handle](#)< [YieldTermStructure](#) > &, const [Date](#) & *newTodaysDate*, const [Date](#) & *newReferenceDate*)

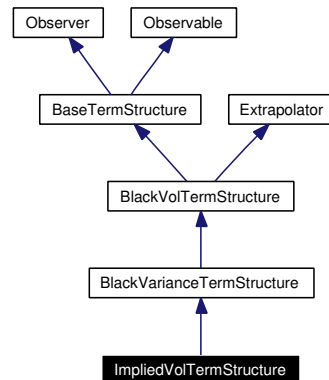
Deprecated

use the constructor without today's date; set the evaluation date through [Settings::instance\(\)](#).

7.257 ImpliedVolTermStructure Class Reference

```
#include <ql/Volatilities/impliedvoltermstructure.hpp>
```

Inheritance diagram for ImpliedVolTermStructure:



7.257.1 Detailed Description

Implied vol term structure at a given date in the future.

The given date will be the implied reference date.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Warning:

It doesn't make financial sense to have an asset-dependant implied Vol Term Structure. This class should be used with term structures that are time dependant only

Public Member Functions

- **ImpliedVolTermStructure** (const [Handle](#)< [BlackVolTermStructure](#) > &originalTS, const [Date](#) &referenceDate)

BlackVolTermStructure interface

- [DayCounter](#) **dayCounter** () const
the day counter used for date/time conversion
- [Date](#) **maxDate** () const
the latest date for which the term structure can return vols
- [Real](#) **minStrike** () const
the minimum strike for which the term structure can return vols
- [Real](#) **maxStrike** () const
the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

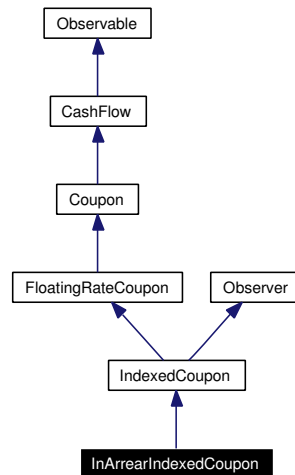
Protected Member Functions

- virtual [Real](#) **blackVarianceImpl** ([Time](#) t, [Real](#) strike) const
Black variance calculation.

7.258 InArrearIndexedCoupon Class Reference

```
#include <ql/CashFlows/inarrearindexedcoupon.hpp>
```

Inheritance diagram for InArrearIndexedCoupon:



7.258.1 Detailed Description

In-arrear floating-rate coupon.

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Tests

The class is tested by comparing the value of an in-arrear swap against a known good value.

Public Member Functions

- **InArrearIndexedCoupon** ([Real](#) nominal, const [Date](#) &paymentDate, const boost::shared_ptr< [Xibor](#) > &index, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())

FloatingRateCoupon interface

- [Date](#) **fixingDate** () const
fixing date

Modifiers

- void **setCapletVolatility** (const [Handle](#)< [CapletVolatilityStructure](#) > &)

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Member Functions

- [RateConvexityAdjustment](#) ([Rate](#) fixing) const
convexity adjustment for the given index fixing

Protected Attributes

- boost::shared_ptr< [Xibor](#) > [xibor_](#)
- [Handle](#)< [CapletVolatilityStructure](#) > [capletVolatility_](#)

7.259 IncrementalStatistics Class Reference

```
#include <ql/Math/incrementalstatistics.hpp>
```

7.259.1 Detailed Description

Statistics tool based on incremental accumulation.

It can accumulate a set of data and return statistics (e.g: mean, variance, skewness, kurtosis, error estimation, etc.)

Warning:

high moments are numerically unstable for high average/standardDeviation ratios

Public Member Functions

Inspectors

- [Size samples](#) () const
number of samples collected
- [Real weightSum](#) () const
sum of data weights
- [Real mean](#) () const
- [Real variance](#) () const
- [Real standardDeviation](#) () const
- [Real downsideVariance](#) () const
- [Real downsideDeviation](#) () const
- [Real errorEstimate](#) () const
- [Real skewness](#) () const
- [Real kurtosis](#) () const
- [Real min](#) () const
- [Real max](#) () const

Modifiers

- void [add](#) ([Real](#) value, [Real](#) weight=1.0)
adds a datum to the set, possibly with a weight
- template<class DataIterator> void [addSequence](#) (DataIterator begin, DataIterator end)
adds a sequence of data to the set, with default weight
- template<class DataIterator, class WeightIterator> void [addSequence](#) (DataIterator begin, DataIterator end, WeightIterator wbegin)
adds a sequence of data to the set, each with its weight
- void [reset](#) ()
resets the data to a null set

Protected Attributes

- [Size](#) sampleNumber_
- [Size](#) downsideSampleNumber_
- [Real](#) sampleWeight_
- [Real](#) downsideSampleWeight_
- [Real](#) sum_
- [Real](#) quadraticSum_
- [Real](#) downsideQuadraticSum_
- [Real](#) cubicSum_
- [Real](#) fourthPowerSum_
- [Real](#) min_
- [Real](#) max_

7.259.2 Member Function Documentation

7.259.2.1 [Real](#) mean () const

returns the mean, defined as

$$\langle x \rangle = \frac{\sum w_i x_i}{\sum w_i}.$$

7.259.2.2 [Real](#) variance () const

returns the variance, defined as

$$\frac{N}{N-1} \langle (x - \langle x \rangle)^2 \rangle.$$

7.259.2.3 [Real](#) standardDeviation () const

returns the standard deviation σ , defined as the square root of the variance.

7.259.2.4 [Real](#) downsideVariance () const

returns the downside variance, defined as

$$\frac{N}{N-1} \times \frac{\sum_{i=1}^N \theta \times x_i^2}{\sum_{i=1}^N w_i}$$

, where $\theta = 0$ if $x > 0$ and $\theta = 1$ if $x < 0$

7.259.2.5 [Real](#) downsideDeviation () const

returns the downside deviation, defined as the square root of the downside variance.

7.259.2.6 Real errorEstimate () const

returns the error estimate ϵ , defined as the square root of the ratio of the variance to the number of samples.

7.259.2.7 Real skewness () const

returns the skewness, defined as

$$\frac{N^2}{(N-1)(N-2)} \frac{\langle (x - \langle x \rangle)^3 \rangle}{\sigma^3}.$$

The above evaluates to 0 for a Gaussian distribution.

7.259.2.8 Real kurtosis () const

returns the excess kurtosis, defined as

$$\frac{N^2(N+1)}{(N-1)(N-2)(N-3)} \frac{\langle (x - \langle x \rangle)^4 \rangle}{\sigma^4} - \frac{3(N-1)^2}{(N-2)(N-3)}.$$

The above evaluates to 0 for a Gaussian distribution.

7.259.2.9 Real min () const

returns the minimum sample value

7.259.2.10 Real max () const

returns the maximum sample value

7.259.2.11 void add (Real value, Real weight = 1.0)

adds a datum to the set, possibly with a weight

Precondition:

weight must be positive or null

7.259.2.12 void addSequence (DataIterator begin, DataIterator end, WeightIterator wbegin)

adds a sequence of data to the set, each with its weight

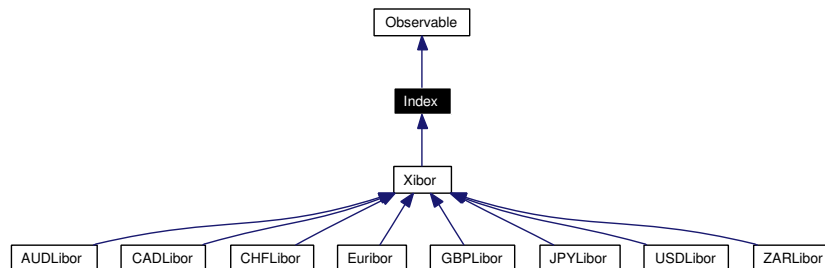
Precondition:

weights must be positive or null

7.260 Index Class Reference

```
#include <ql/index.hpp>
```

Inheritance diagram for Index:



7.260.1 Detailed Description

purely virtual base class for indexes

Public Member Functions

- virtual `std::string name () const =0`
Returns the name of the index.
- virtual `Rate fixing (const Date &fixingDate) const =0`
returns the fixing at the given date

7.260.2 Member Function Documentation

7.260.2.1 virtual `std::string name () const` [pure virtual]

Returns the name of the index.

Warning:

This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implemented in [Xibor](#).

7.260.2.2 virtual `Rate fixing (const Date &fixingDate) const` [pure virtual]

returns the fixing at the given date

Note:

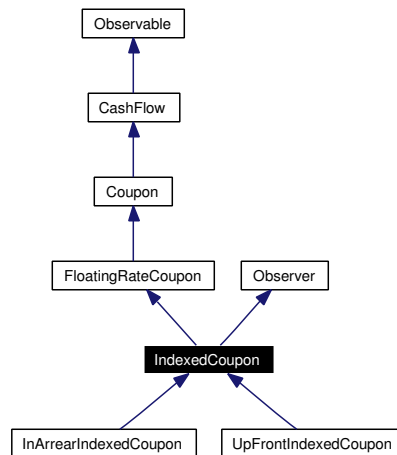
any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implemented in [Xibor](#).

7.261 IndexedCoupon Class Reference

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

Inheritance diagram for IndexedCoupon:



7.261.1 Detailed Description

Base indexed coupon class.

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **IndexedCoupon** ([Real](#) nominal, const [Date](#) &paymentDate, const boost::shared_ptr< [Index](#) > &index, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())

CashFlow interface

- [Real](#) **amount** () const
returns the amount of the cash flow

Coupon interface

- [DayCounter](#) **dayCounter** () const
day counter for accrual calculation

FloatingRateCoupon interface

- [Rate](#) **indexFixing** () const

fixing of the underlying index

- [Rate fixing](#) () const

Inspectors

- const boost::shared_ptr< [Index](#) > & [index](#) () const

Observer interface

- void [update](#) ()

Visitability

- virtual void [accept](#) ([AcyclicVisitor](#) &)

7.261.2 Member Function Documentation

7.261.2.1 [Real](#) amount () const [virtual]

returns the amount of the cash flow

Note:

The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

7.261.2.2 [Rate fixing](#) () const [virtual]

Deprecated

use [rate\(\)](#) instead

Implements [FloatingRateCoupon](#).

7.261.2.3 void [update](#) () [virtual]

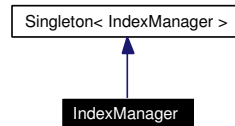
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.262 IndexManager Class Reference

```
#include <ql/Indexes/indexmanager.hpp>
```

Inheritance diagram for IndexManager:



7.262.1 Detailed Description

global repository for past index fixings

Public Member Functions

- void **setHistory** (const std::string &name, const [History](#) &)
- const [History](#) & **getHistory** (const std::string &name) const
- bool **hasHistory** (const std::string &name) const
- std::vector< std::string > **histories** () const

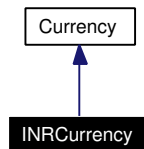
Friends

- class `Singleton<IndexManager>`

7.263 INRCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for INRCurrency:



7.263.1 Detailed Description

Indian rupee.

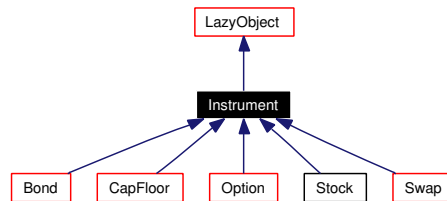
The ISO three-letter code is INR; the numeric code is 356. It is divided in 100 paise.

ingroup currencies

7.264 Instrument Class Reference

```
#include <ql/instrument.hpp>
```

Inheritance diagram for Instrument:



7.264.1 Detailed Description

Abstract instrument class.

This class is purely abstract and defines the interface of concrete instruments which will be derived from this one.

Tests

observability of class instances is checked.

Public Member Functions

- virtual void [setupArguments](#) ([Arguments](#) *) const

Inspectors

- [Real NPV](#) () const
returns the net present value of the instrument.
- [Real errorEstimate](#) () const
returns the error estimate on the NPV when available.
- virtual bool [isExpired](#) () const =0
returns whether the instrument is still tradable.

Modifiers

- void [setPricingEngine](#) (const boost::shared_ptr< [PricingEngine](#) > &)
set the pricing engine to be used.

Protected Member Functions

Calculations

- void [calculate](#) () const
- virtual void [setupExpired](#) () const
- virtual void [performCalculations](#) () const

Protected Attributes

- `boost::shared_ptr< PricingEngine > engine_`

Results

The value of this attribute and any other that derived classes might declare must be set during calculation.

- `Real NPV_`
- `Real errorEstimate_`

7.264.2 Member Function Documentation

7.264.2.1 `void setPricingEngine (const boost::shared_ptr< PricingEngine > &)`

set the pricing engine to be used.

Warning:

calling this method will have no effects in case the **performCalculation** method was overridden in a derived class.

7.264.2.2 `void setupArguments (Arguments *) const [virtual]`

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented in [ContinuousAveragingAsianOption](#), [DiscreteAveragingAsianOption](#), [BarrierOption](#), [BasketOption](#), [CapFloor](#), [CliquetOption](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoForwardVanillaOption](#), [QuantoVanillaOption](#), [SimpleSwap](#), and [Swaption](#).

7.264.2.3 `void calculate () const [protected, virtual]`

This method performs all needed calculations by calling the **performCalculations** method.

Warning:

Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.

Should this method be redefined in derived classes, [LazyObject::calculate\(\)](#) should be called in the overriding method.

Reimplemented from [LazyObject](#).

7.264.2.4 `void setupExpired () const [protected, virtual]`

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented in [MultiAssetOption](#), [OneAssetOption](#), [QuantoVanillaOption](#), and [Swap](#).

7.264.2.5 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Implements [LazyObject](#).

Reimplemented in [BarrierOption](#), [Bond](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), [Stock](#), and [Swap](#).

7.265 IntegerFormatter Class Reference

```
#include <ql/basicdataformatters.hpp>
```

7.265.1 Detailed Description

Formats integers for output.

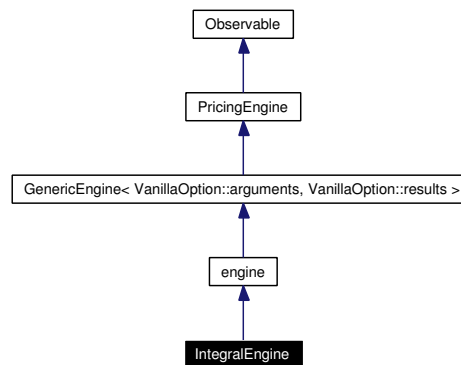
Static Public Member Functions

- `std::string toString` ([BigInteger](#) l, [Integer](#) digits=0)
- `std::string toPowerOfTwo` ([BigInteger](#) l, [Integer](#) digits=0)

7.266 IntegralEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/integralengine.hpp>
```

Inheritance diagram for IntegralEngine:



7.266.1 Detailed Description

Pricing engine for European vanilla options using integral approach

Todo

define tolerance for calculate()

Public Member Functions

- void **calculate** () const

7.267 InterestRate Class Reference

```
#include <ql/interestrate.hpp>
```

7.267.1 Detailed Description

Concrete interest rate class.

This class encapsulate the interest rate compounding algebra. It manages day-counting conventions, compounding conventions, conversion between different conventions, discount/compound factor calculations, and implied/equivalent rate calculations.

Tests

Converted rates are checked against known good results

Public Member Functions

constructors

- [InterestRate](#) ()
Default constructor returning a null interest rate.
- [InterestRate](#) ([Rate](#) r, const [DayCounter](#) &dc, Compounding comp, [Frequency](#) freq=[Annual](#))
Standard constructor.

conversions

- **operator Rate** () const

inspectors

- [Rate](#) **rate** () const
- const [DayCounter](#) & **dayCounter** () const
- Compounding **compounding** () const
- [Frequency](#) **frequency** () const

discount/compound factor calculations

- [DiscountFactor](#) **discountFactor** ([Time](#) t) const
discount factor implied by the rate compounded at time t.
- [DiscountFactor](#) **discountFactor** (const [Date](#) &d1, const [Date](#) &d2) const
discount factor implied by the rate compounded between two dates
- [Real compoundFactor](#) ([Time](#) t) const
compound factor implied by the rate compounded at time t.
- [Real compoundFactor](#) (const [Date](#) &d1, const [Date](#) &d2) const
compound factor implied by the rate compounded between two dates

equivalent rate calculations

- [InterestRate](#) [equivalentRate](#) ([Time](#) t, [Compounding](#) comp, [Frequency](#) freq=[Annual](#)) const
equivalent interest rate for a compounding period t.
- [InterestRate](#) [equivalentRate](#) ([Date](#) d1, [Date](#) d2, const [DayCounter](#) &resultDC, [Compounding](#) comp, [Frequency](#) freq=[Annual](#)) const
equivalent rate for a compounding period between two dates

Static Public Member Functions

implied rate calculations

- [InterestRate](#) [impliedRate](#) ([Real](#) compound, [Time](#) t, const [DayCounter](#) &resultDC, [Compounding](#) comp, [Frequency](#) freq=[Annual](#))
implied interest rate for a given compound factor at a given time.
- [InterestRate](#) [impliedRate](#) ([Real](#) compound, const [Date](#) &d1, const [Date](#) &d2, const [DayCounter](#) &resultDC, [Compounding](#) comp, [Frequency](#) freq=[Annual](#))
implied rate for a given compound factor between two dates.

7.267.2 Member Function Documentation

7.267.2.1 [DiscountFactor](#) discountFactor ([Time](#) t) const

discount factor implied by the rate compounded at time t.

Warning:

Time must be measured using [InterestRate](#)'s own day counter.

7.267.2.2 [Real](#) compoundFactor ([Time](#) t) const

compound factor implied by the rate compounded at time t.

returns the compound (a.k.a capitalization) factor implied by the rate compounded at time t.

Warning:

Time must be measured using [InterestRate](#)'s own day counter.

7.267.2.3 [Real](#) compoundFactor (const [Date](#) & d1, const [Date](#) & d2) const

compound factor implied by the rate compounded between two dates

returns the compound (a.k.a capitalization) factor implied by the rate compounded between two dates.

7.267.2.4 **InterestRate** `impliedRate (Real compound, Time t, const DayCounter & resultDC, Compounding comp, Frequency freq = Annual)` [static]

implied interest rate for a given compound factor at a given time.

The resulting **InterestRate** has the day-counter provided as input.

Warning:

Time must be measured using the day-counter provided as input.

7.267.2.5 **InterestRate** `impliedRate (Real compound, const Date & d1, const Date & d2, const DayCounter & resultDC, Compounding comp, Frequency freq = Annual)` [static]

implied rate for a given compound factor between two dates.

The resulting rate is calculated taking the required day-counting rule into account.

7.267.2.6 **InterestRate** `equivalentRate (Time t, Compounding comp, Frequency freq = Annual)` const

equivalent interest rate for a compounding period t.

The resulting **InterestRate** shares the same implicit day-counting rule of the original **InterestRate** instance.

Warning:

Time must be measured using the **InterestRate**'s own day counter.

7.267.2.7 **InterestRate** `equivalentRate (Date d1, Date d2, const DayCounter & resultDC, Compounding comp, Frequency freq = Annual)` const

equivalent rate for a compounding period between two dates

The resulting rate is calculated taking the required day-counting rule into account.

7.268 InterestRateFormatter Class Reference

```
#include <ql/interestrates.hpp>
```

7.268.1 Detailed Description

Formats interest rates for output.

Combines [RateFormatter](#) and [CompoundingRuleFormatter](#) with information about the day-counting convention

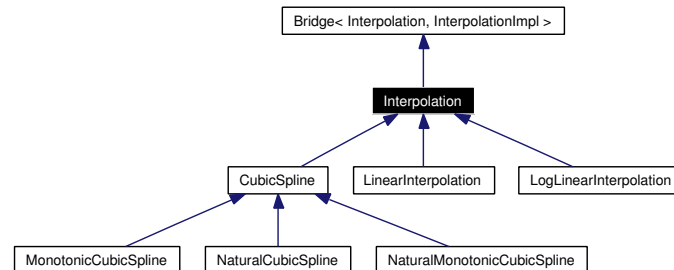
Static Public Member Functions

- `std::string toString` ([InterestRate](#) ir, [Integer](#) precision=5)

7.269 Interpolation Class Reference

```
#include <ql/Math/interpolation.hpp>
```

Inheritance diagram for Interpolation:



7.269.1 Detailed Description

base class for 1-D interpolations.

Classes derived from this class will provide interpolated values from two sequences of equal length, representing discretized values of a variable and a function of the former, respectively.

Public Types

- typedef [Real](#) **argument_type**
- typedef [Real](#) **result_type**

Public Member Functions

- [Real](#) **operator()** ([Real](#) x, bool allowExtrapolation=false) const
- [Real](#) **primitive** ([Real](#) x, bool allowExtrapolation=false) const
- [Real](#) **derivative** ([Real](#) x, bool allowExtrapolation=false) const
- [Real](#) **secondDerivative** ([Real](#) x, bool allowExtrapolation=false) const
- [Real](#) **xMin** () const
- [Real](#) **xMax** () const
- bool **isInRange** ([Real](#) x) const

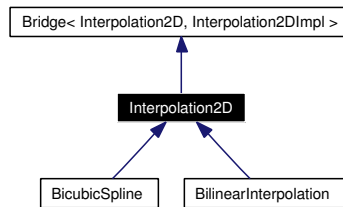
Protected Member Functions

- void **checkRange** ([Real](#) x, bool allowExtrapolation) const

7.270 Interpolation2D Class Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2D:



7.270.1 Detailed Description

base class for 2-D interpolations.

Classes derived from this class will provide interpolated values from two sequences of length N and M , representing the discretized values of the x and y variables, and a $N \times M$ matrix representing the tabulated function values.

Public Types

- typedef [Real](#) `first_argument_type`
- typedef [Real](#) `second_argument_type`
- typedef [Real](#) `result_type`

Public Member Functions

- [Real](#) `operator()` ([Real](#) x , [Real](#) y , bool `allowExtrapolation=false`) const
- [Real](#) `xMin` () const
- [Real](#) `xMax` () const
- [Real](#) `yMin` () const
- [Real](#) `yMax` () const
- bool `isInRange` ([Real](#) x , [Real](#) y) const

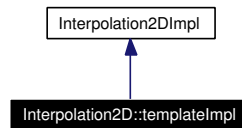
Protected Member Functions

- void `checkRange` ([Real](#) x , [Real](#) y , bool `allowExtrapolation`) const

7.271 Interpolation2D::templateImpl Class Template Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2D::templateImpl:



7.271.1 Detailed Description

```
template<class I1, class I2, class M> class QuantLib::Interpolation2D::templateImpl< I1, I2, M  
>
```

basic template implementation

Public Member Functions

- **templateImpl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, const I2 &yEnd, const M &zData)
- [Real](#) xMin () const
- [Real](#) xMax () const
- [Real](#) yMin () const
- [Real](#) yMax () const
- bool **isInRange** ([Real](#) x, [Real](#) y) const

Protected Member Functions

- [Size](#) locateX ([Real](#) x) const
- [Size](#) locateY ([Real](#) y) const

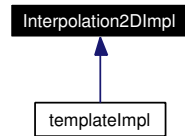
Protected Attributes

- I1 xBegin_
- I1 xEnd_
- I2 yBegin_
- I2 yEnd_
- const M & zData_

7.272 Interpolation2DImpl Class Reference

```
#include <ql/Math/interpolation2D.hpp>
```

Inheritance diagram for Interpolation2DImpl:



7.272.1 Detailed Description

abstract base class for 2-D interpolation implementations

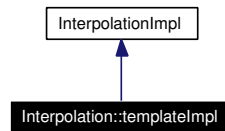
Public Member Functions

- virtual [Real](#) **xMin** () const =0
- virtual [Real](#) **xMax** () const =0
- virtual [Real](#) **yMin** () const =0
- virtual [Real](#) **yMax** () const =0
- virtual bool **isInRange** ([Real](#) x, [Real](#) y) const =0
- virtual [Real](#) **value** ([Real](#) x, [Real](#) y) const =0

7.273 Interpolation::templateImpl Class Template Reference

```
#include <ql/Math/interpolation.hpp>
```

Inheritance diagram for Interpolation::templateImpl:



7.273.1 Detailed Description

```
template<class I1, class I2> class QuantLib::Interpolation::templateImpl< I1, I2 >
```

basic template implementation

Public Member Functions

- **templateImpl** (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)
- [Real](#) **xMin** () const
- [Real](#) **xMax** () const
- bool **isInRange** ([Real](#) x) const

Protected Member Functions

- [Size](#) **locate** ([Real](#) x) const

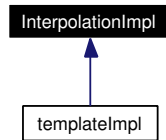
Protected Attributes

- I1 **xBegin_**
- I1 **xEnd_**
- I2 **yBegin_**

7.274 InterpolationImpl Class Reference

```
#include <ql/Math/interpolation.hpp>
```

Inheritance diagram for InterpolationImpl:



7.274.1 Detailed Description

abstract base class for interpolation implementations

Public Member Functions

- virtual [Real](#) **xMin** () const =0
- virtual [Real](#) **xMax** () const =0
- virtual bool **isInRange** ([Real](#)) const =0
- virtual [Real](#) **value** ([Real](#)) const =0
- virtual [Real](#) **primitive** ([Real](#)) const =0
- virtual [Real](#) **derivative** ([Real](#)) const =0
- virtual [Real](#) **secondDerivative** ([Real](#)) const =0

7.275 InverseCumulativeNormal Class Reference

```
#include <ql/Math/normaldistribution.hpp>
```

7.275.1 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](http://home.online.no/~pjacklam/notes/invnorm/index.html), Statistics Division. URL: <http://home.online.no/~pjacklam/notes/invnorm/index.html>

This class can also be used to generate a gaussian normal distribution from a uniform distribution. This is especially useful when a gaussian normal distribution is generated from a low discrepancy uniform distribution: in this case the traditional Box-Muller approach and its variants would not preserve the sequence's low-discrepancy.

Public Member Functions

- **InverseCumulativeNormal** ([Real](#) average=0.0, [Real](#) sigma=1.0)
- **Real operator()** ([Real](#) x) const

7.276 InverseCumulativePoisson Class Reference

```
#include <ql/Math/poissondistribution.hpp>
```

7.276.1 Detailed Description

Inverse cumulative Poisson distribution function.

Tests

the correctness of the returned value is tested by checking it against known good results.

Public Member Functions

- **InverseCumulativePoisson** ([Real](#) lambda=1.0)
- **Real operator()** ([Real](#) x) const

7.277 InverseCumulativeRng Class Template Reference

```
#include <ql/RandomNumbers/inversecumulativrng.hpp>
```

7.277.1 Detailed Description

```
template<class RNG, class IC> class QuantLib::InverseCumulativeRng< RNG, IC >
```

Inverse cumulative random number generator.

It uses a uniform deviate in (0, 1) as the source of cumulative distribution values. Then an inverse cumulative distribution is used to calculate the distribution deviate.

The uniform deviate is supplied by RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

The inverse cumulative distribution is supplied by IC.

Class IC must implement the following interface:

```
IC::IC();  
Real IC::operator() const;
```

Public Types

- typedef [Sample](#)< [Real](#) > **sample_type**
- typedef RNG **urng_type**

Public Member Functions

- **InverseCumulativeRng** (const RNG &uniformGenerator)
- [sample_type](#) **next** () const
returns a sample from a Gaussian distribution

7.278 InverseCumulativeRsg Class Template Reference

```
#include <ql/RandomNumbers/inversecumulativersg.hpp>
```

7.278.1 Detailed Description

```
template<class USG, class IC> class QuantLib::InverseCumulativeRsg< USG, IC >
```

Inverse cumulative random sequence generator.

It uses a sequence of uniform deviate in (0, 1) as the source of cumulative distribution values. Then an inverse cumulative distribution is used to calculate the distribution deviate.

The uniform deviate sequence is supplied by USG.

Class USG must implement the following interface:

```
USG::sample_type USG::nextSequence() const;
Size USG::dimension() const;
```

The inverse cumulative distribution is supplied by IC.

Class IC must implement the following interface:

```
IC::IC();
Real IC::operator() const;
```

Public Types

- typedef [Sample< Array >](#) **sample_type**

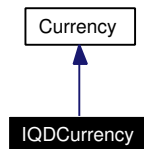
Public Member Functions

- **InverseCumulativeRsg** (const USG &uniformSequenceGenerator)
- **InverseCumulativeRsg** (const USG &uniformSequenceGenerator, const IC &inverseCumulative)
- const [sample_type](#) & [nextSequence](#) () const
returns next sample from the Gaussian distribution
- const [sample_type](#) & [lastSequence](#) () const
- [Size](#) [dimension](#) () const

7.279 IQDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for IQDCurrency:



7.279.1 Detailed Description

Iraqi dinar.

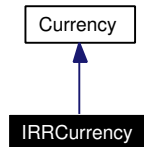
The ISO three-letter code is IQD; the numeric code is 368. It is divided in 1000 fils.

ingroup currencies

7.280 IRRCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for IRRCurrency:



7.280.1 Detailed Description

Iranian rial.

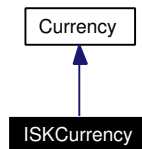
The ISO three-letter code is IRR; the numeric code is 364. It has no subdivisions.

ingroup currencies

7.281 ISKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ISKCurrency:



7.281.1 Detailed Description

Iceland krona.

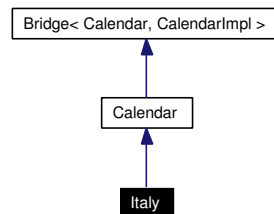
The ISO three-letter code is ISK; the numeric code is 352. It is divided in 100 aurar.

ingroup currencies

7.282 Italy Class Reference

```
#include <ql/Calendars/italy.hpp>
```

Inheritance diagram for Italy:



7.282.1 Detailed Description

Italian calendars.

Public holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Epiphany, January 6th
- Easter Monday
- Liberation Day, April 25th
- Labour Day, May 1st
- Republic Day, June 2nd (since 2000)
- Assumption, August 15th
- All Saint's Day, November 1st
- Immaculate Conception Day, December 8th
- Christmas Day, December 25th
- St. Stephen's Day, December 26th

Holidays for the stock exchange (data from <http://www.borsaitalia.it>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday
- Easter Monday

- Labour Day, May 1st
- Assumption, August 15th
- Christmas' Eve, December 24th
- Christmas, December 25th
- St. Stephen, December 26th
- New Year's Eve, December 31st

Tests

the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum [Market](#) { [Settlement](#), [Exchange](#) }
Italian calendars.

Public Member Functions

- Italy ([Market](#) market=Settlement)

7.282.2 Member Enumeration Documentation

7.282.2.1 enum [Market](#)

Italian calendars.

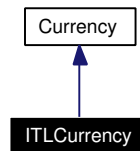
Enumeration values:

- Settlement* generic settlement calendar
- Exchange* Milan stock-exchange calendar.

7.283 ITLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ITLCurrency:



7.283.1 Detailed Description

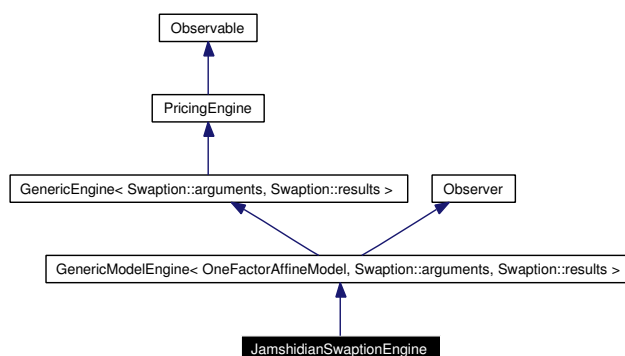
Italian lira.

The ISO three-letter code was ITL; the numeric code was 380. It had no subdivisions.

7.284 JamshidianSwaptionEngine Class Reference

```
#include <ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp>
```

Inheritance diagram for JamshidianSwaptionEngine:



7.284.1 Detailed Description

Jamshidian swaption engine.

Public Member Functions

- **JamshidianSwaptionEngine** (const boost::shared_ptr< [OneFactorAffineModel](#) > &model)
- void **calculate** () const

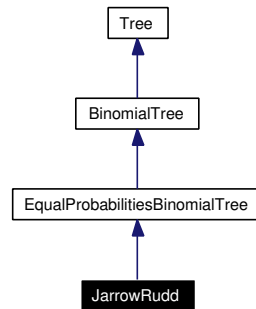
Friends

- class **rStarFinder**

7.285 JarrowRudd Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for JarrowRudd:



7.285.1 Detailed Description

Jarrow-Rudd (multiplicative) equal probabilities binomial tree.

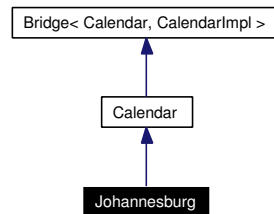
Public Member Functions

- **JarrowRudd** (const boost::shared_ptr< [StochasticProcess](#) > &process, [Time](#) end, [Size](#) steps, [Real](#) strike)

7.286 Johannesburg Class Reference

```
#include <ql/Calendars/johannesburg.hpp>
```

Inheritance diagram for Johannesburg:



7.286.1 Detailed Description

Johannesburg calendar

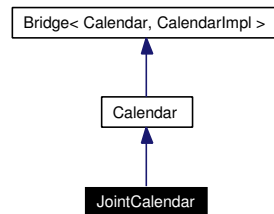
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Family Day, Easter Monday
- Human Rights Day, March 21st (possibly moved to Monday)
- Freedom Day, April 27th (possibly moved to Monday)
- Workers Day, May 1st (possibly moved to Monday)
- Youth Day, June 16th (possibly moved to Monday)
- National Women's Day, August 9th (possibly moved to Monday)
- Heritage Day, September 24th (possibly moved to Monday)
- Day of Reconciliation, December 16th (possibly moved to Monday)
- Christmas December 25th
- Day of Goodwill December 26th (possibly moved to Monday)

7.287 JointCalendar Class Reference

```
#include <ql/Calendars/jointcalendar.hpp>
```

Inheritance diagram for JointCalendar:



7.287.1 Detailed Description

Joint calendar.

Depending on the chosen rule, this calendar has a set of business days given by either the union or the intersection of the sets of business days of the given calendars.

Tests

the correctness of the returned results is tested by reproducing the calculations.

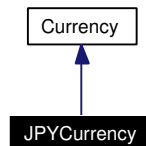
Public Member Functions

- **JointCalendar** (const [Calendar](#) &, const [Calendar](#) &, JointCalendarRule=JoinHolidays)
- **JointCalendar** (const [Calendar](#) &, const [Calendar](#) &, const [Calendar](#) &, JointCalendarRule=JoinHolidays)
- **JointCalendar** (const [Calendar](#) &, const [Calendar](#) &, const [Calendar](#) &, const [Calendar](#) &, JointCalendarRule=JoinHolidays)

7.288 JPYCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for JPYCurrency:



7.288.1 Detailed Description

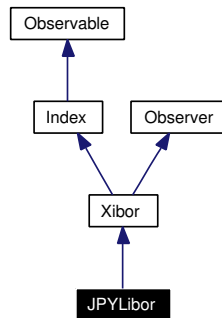
Japanese yen.

The ISO three-letter code is JPY; the numeric code is 392. It is divided into 100 sen.

7.289 JPYLibor Class Reference

```
#include <ql/Indexes/jpylibor.hpp>
```

Inheritance diagram for JPYLibor:



7.289.1 Detailed Description

JPY Libor index, also known as TIBOR

Todo

check settlement days

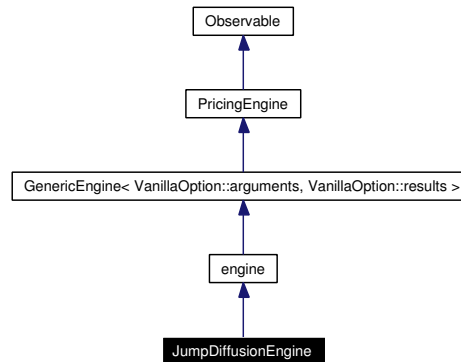
Public Member Functions

- **JPYLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

7.290 JumpDiffusionEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp>
```

Inheritance diagram for JumpDiffusionEngine:



7.290.1 Detailed Description

Jump-diffusion engine for vanilla options.

Tests

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

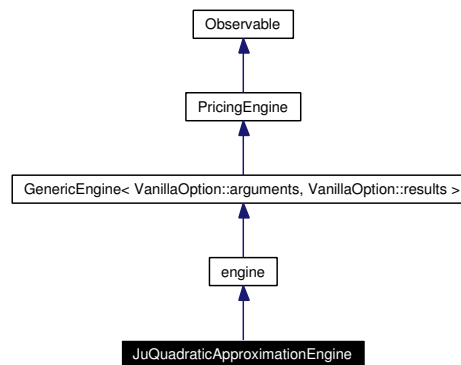
Public Member Functions

- **JumpDiffusionEngine** (const boost::shared_ptr< [VanillaOption::engine](#) > &, [Real](#) relative-Accuracy_=1e-4, [Size](#) maxIterations=100)
- void **calculate** () const

7.291 JuQuadraticApproximationEngine Class Reference

```
#include <ql/PricingEngines/Vanilla/juquadraticengine.hpp>
```

Inheritance diagram for JuQuadraticApproximationEngine:



7.291.1 Detailed Description

Pricing engine for American options with Ju quadratic approximation

An Approximate Formula for Pricing American Options Journal of Derivatives Winter 1999 Ju, N.

Warning:

Barone-Adesi-Whaley critical commodity price calculation is used, it has not been modified to see whether the method of Ju is faster. Ju does not say how he solves the equation for the critical stock price, e.g. [Newton](#) method. He just gives the solution. The method of BAW gives answers to the same accuracy as in Ju (1999)

Tests

the correctness of the returned value is tested by reproducing results available in literature.

Bug

test fails for Borland compiler

Public Member Functions

- void **calculate** () const

7.292 KnuthUniformRng Class Reference

```
#include <ql/RandomNumbers/knuthuniformrng.hpp>
```

7.292.1 Detailed Description

Uniform random number generator.

Random number generator by Knuth. For more details see Knuth, Seminumerical Algorithms, 3rd edition, Section 3.6.

Note:

This is **not** Knuth's original implementation which is available at <http://www-cs-faculty.stanford.edu/~knuth/programs.html>, but rather a slightly modified version wrapped in a C++ class. Such modifications did not affect the code but only the data structures used, which were converted to their standard C++ equivalents.

Public Types

- typedef [Sample](#)< [Real](#) > `sample_type`

Public Member Functions

- [KnuthUniformRng](#) (long seed=0)
- [sample_type next](#) () const

7.292.2 Constructor & Destructor Documentation

7.292.2.1 [KnuthUniformRng](#) (long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.292.3 Member Function Documentation

7.292.3.1 [KnuthUniformRng::sample_type next](#) () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

7.293 KronrodIntegral Class Reference

```
#include <ql/Math/kronrodintegral.hpp>
```

7.293.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

References:

Gauss-Kronrod Integration <<http://mathcssun1.emporia.edu/~oneilcat/Experiment-Applet3/ExperimentApplet3.html>>

NMS - Numerical Analysis Library <http://www.math.iastate.edu/burkardt/f_src/nms/nms.html>

Tests

the correctness of the result is tested by checking it against known good values.

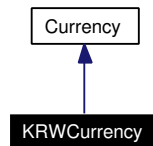
Public Member Functions

- **KronrodIntegral** ([Real](#) tolerance, [Size](#) maxFunctionEvaluations=[Null](#)< [Size](#) >())
- `template<class F> Real operator()` (const F &f, [Real](#) a, [Real](#) b) const
- [Size](#) functionEvaluations ()

7.294 KRWCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for KRWCurrency:



7.294.1 Detailed Description

South-Korean won.

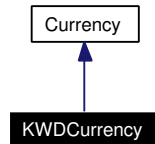
The ISO three-letter code is KRW; the numeric code is 410. It is divided in 100 chon.

ingroup currencies

7.295 KWDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for KWDCurrency:



7.295.1 Detailed Description

Kuwaiti dinar.

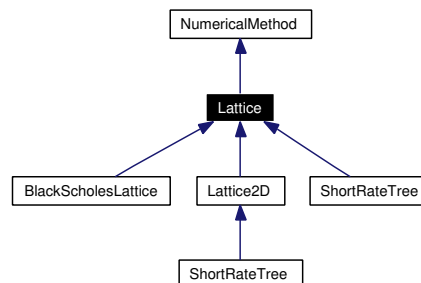
The ISO three-letter code is KWD; the numeric code is 414. It is divided in 1000 fils.

ingroup currencies

7.296 Lattice Class Reference

```
#include <ql/Lattices/lattice.hpp>
```

Inheritance diagram for Lattice:



7.296.1 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)
- 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 properties.
- virtual [Real](#) **probability** ([Size](#) i, [Size](#) index, [Size](#) branch) const =0

NumericalMethod interface

- void **initialize** ([DiscretizedAsset](#) &, [Time](#) t) const
initialize an asset at the given time.
- void **rollback** ([DiscretizedAsset](#) &, [Time](#) to) const
- void **partialRollback** ([DiscretizedAsset](#) &, [Time](#) to) const
- [Real](#) **presentValue** ([DiscretizedAsset](#) &)
Computes the present value of an asset using Arrow-Debreu prices.

Protected Member Functions

- void **computeStatePrices** ([Size](#) until)
- virtual void **stepback** ([Size](#) i, const [Array](#) &values, [Array](#) &newValues) const

Protected Attributes

- `std::vector< Array > statePrices_`

7.296.2 Member Function Documentation

7.296.2.1 `void rollback (DiscretizedAsset &, Time to) const` [virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implements [NumericalMethod](#).

7.296.2.2 `void partialRollback (DiscretizedAsset &, Time to) const` [virtual]

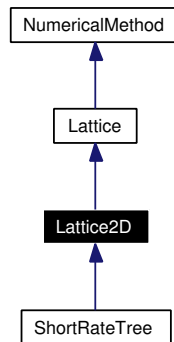
Roll back an asset until the given time, but do not perform the final adjustment.

Implements [NumericalMethod](#).

7.297 Lattice2D Class Reference

```
#include <ql/Lattices/lattice2d.hpp>
```

Inheritance diagram for Lattice2D:



7.297.1 Detailed Description

Two-dimensional lattice.

This lattice is based on two trinomial trees and primarily used for the [G2](#) short-rate model.

Public Member Functions

- **Lattice2D** (const boost::shared_ptr< [TrinomialTree](#) > &tree1, const boost::shared_ptr< [TrinomialTree](#) > &tree2, [Real](#) correlation)
- **Size size** ([Size](#) i) const

Protected Member Functions

- **Size descendant** ([Size](#) i, [Size](#) index, [Size](#) branch) const
Tree properties.
- **Real probability** ([Size](#) i, [Size](#) index, [Size](#) branch) const

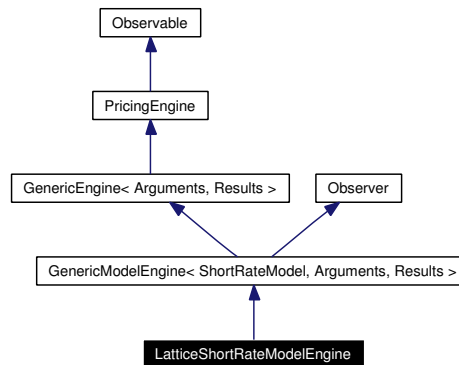
Protected Attributes

- boost::shared_ptr< [Tree](#) > **tree1_**
- boost::shared_ptr< [Tree](#) > **tree2_**

7.298 LatticeShortRateModelEngine Class Template Reference

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

Inheritance diagram for LatticeShortRateModelEngine:



7.298.1 Detailed Description

template<class Arguments, class Results> class QuantLib::LatticeShortRateModelEngine< Arguments, Results >

Engine for a short-rate model specialized on a lattice.

Derived engines only need to implement the `calculate()` method

Public Member Functions

- **LatticeShortRateModelEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, [Size](#) timeSteps)
- **LatticeShortRateModelEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, const [TimeGrid](#) &timeGrid)
- void [update](#) ()

Protected Attributes

- [TimeGrid](#) timeGrid_
- [Size](#) timeSteps_
- boost::shared_ptr< [Lattice](#) > lattice_

7.298.2 Member Function Documentation

7.298.2.1 void update () [virtual]

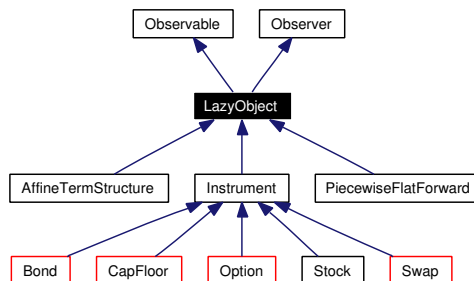
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [GenericModelEngine< ShortRateModel, Arguments, Results >](#).

7.299 LazyObject Class Reference

```
#include <ql/Patterns/lazyobject.hpp>
```

Inheritance diagram for LazyObject:



7.299.1 Detailed Description

Framework for calculation on demand and result caching.

Calculations

These methods do not modify the structure of the object and are therefore declared as `const`. Data members which will be calculated on demand need to be declared as mutable.

- void `recalculate` ()
- void `freeze` ()
- void `unfreeze` ()
- virtual void `calculate` () `const`
- virtual void `performCalculations` () `const` =0

Public Member Functions

Observer interface

- void `update` ()

Protected Attributes

- bool `calculated_`
- bool `frozen_`

7.299.2 Member Function Documentation

7.299.2.1 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

Reimplemented in [AffineTermStructure](#), and [PiecewiseFlatForward](#).

7.299.2.2 void recalculate ()

This method force the recalculation of any results which would otherwise be cached. It is not declared as const since it needs to call the non-const **notifyObservers** method.

Note:

Explicit invocation of this method is **not** necessary if the object registered itself as observer with the structures on which such results depend. It is strongly advised to follow this policy when possible.

7.299.2.3 void freeze ()

This method constrains the object to return the presently cached results on successive invocations, even if arguments upon which they depend should change.

7.299.2.4 void unfreeze ()

This method reverts the effect of the **freeze** method, thus re-enabling recalculations.

7.299.2.5 void calculate () const [protected, virtual]

This method performs all needed calculations by calling the **performCalculations** method.

Warning:

Objects cache the results of the previous calculation. Such results will be returned upon later invocations of **calculate**. When the results depend on arguments which could change between invocations, the lazy object must register itself as observer of such objects for the calculations to be performed again when they change.

Should this method be redefined in derived classes, [LazyObject::calculate\(\)](#) should be called in the overriding method.

Reimplemented in [Instrument](#).

7.299.2.6 virtual void performCalculations () const [protected, pure virtual]

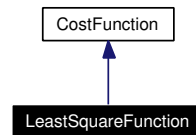
This method must implement any calculations which must be (re)done in order to calculate the desired results.

Implemented in [Instrument](#), [BarrierOption](#), [Bond](#), [ForwardVanillaOption](#), [MultiAssetOption](#), [OneAssetOption](#), [OneAssetStrikedOption](#), [QuantoVanillaOption](#), [Stock](#), and [Swap](#).

7.300 LeastSquareFunction Class Reference

```
#include <ql/Optimization/leastsquare.hpp>
```

Inheritance diagram for LeastSquareFunction:



7.300.1 Detailed Description

Cost function for least-square problems.

Implements a cost function using the interface provided by the [LeastSquareProblem](#) class.

Public Member Functions

- [LeastSquareFunction](#) ([LeastSquareProblem](#) &lsp)
Default constructor.
- virtual [~LeastSquareFunction](#) ()
Destructor.
- virtual [Real value](#) (const [Array](#) &x) const
compute value of the least square function
- virtual void [gradient](#) ([Array](#) &grad_f, const [Array](#) &x) const
compute vector of derivatives of the least square function
- virtual [Real valueAndGradient](#) ([Array](#) &grad_f, const [Array](#) &x) const
compute value and gradient of the least square function

Protected Attributes

- [LeastSquareProblem](#) & lsp_
least square problem

7.301 LeastSquareProblem Class Reference

```
#include <ql/Optimization/leastsquare.hpp>
```

7.301.1 Detailed Description

Base class for least square problem.

Public Member Functions

- virtual [Size](#) [size](#) ()=0
size of the problem ie size of target vector
- virtual void [targetAndValue](#) (const [Array](#) &x, [Array](#) &target, [Array](#) &fct2fit)=0
compute the target vector and the values of the function to fit
- virtual void [targetValueAndGradient](#) (const [Array](#) &x, [Matrix](#) &grad_fct2fit, [Array](#) &target, [Array](#) &fct2fit)=0

7.301.2 Member Function Documentation

7.301.2.1 virtual void [targetValueAndGradient](#) (const [Array](#) & x, [Matrix](#) & grad_fct2fit, [Array](#) & target, [Array](#) & fct2fit) [pure virtual]

compute the target vector, the values of the function to fit and the matrix of derivatives

7.302 LecuyerUniformRng Class Reference

```
#include <ql/RandomNumbers/lecuyeruniformrng.hpp>
```

7.302.1 Detailed Description

Uniform random number generator.

Random number generator of L'Ecuyer with added Bays-Durham shuffle (know as ran2 in Numerical recipes)

For more details see Section 7.1 of Numerical Recipes in C, 2nd Edition, Cambridge University Press (available at <http://www.nr.com/>)

Public Types

- typedef [Sample< Real >](#) `sample_type`

Public Member Functions

- [LecuyerUniformRng](#) (long seed=0)
- [sample_type next](#) () const

7.302.2 Constructor & Destructor Documentation

7.302.2.1 [LecuyerUniformRng](#) (long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.302.3 Member Function Documentation

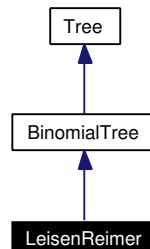
7.302.3.1 [sample_type next](#) () const

returns a sample with weight 1.0 containing a random number uniformly chosen from (0.0,1.0)

7.303 LeisenReimer Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for LeisenReimer:



7.303.1 Detailed Description

Leisen & Reimer tree: multiplicative approach.

Public Member Functions

- **LeisenReimer** (const boost::shared_ptr< [StochasticProcess](#) > &process, [Time](#) end, [Size](#) steps, [Real](#) strike)
- [Real](#) underlying ([Size](#) i, [Size](#) index) const
- [Real](#) probability ([Size](#), [Size](#), [Size](#)) const

Protected Attributes

- [Real](#) up_
- [Real](#) down_
- [Real](#) pu_
- [Real](#) pd_

7.304 LexicographicalView Class Template Reference

```
#include <ql/Math/lexicographicalview.hpp>
```

7.304.1 Detailed Description

template<class RandomAccessIterator> class QuantLib::LexicographicalView< RandomAccessIterator >

Lexicographical 2-D view of a contiguous set of data.

This view can be used to easily store a discretized 2-D function in an array to be used in a finite differences calculation.

Public Types

- typedef RandomAccessIterator [x_iterator](#)
iterates over v_{ij} with j fixed.
- typedef boost::reverse_iterator< RandomAccessIterator > [reverse_x_iterator](#)
iterates backwards over v_{ij} with j fixed.
- typedef [step_iterator](#)< RandomAccessIterator > [y_iterator](#)
iterates over v_{ij} with i fixed.
- typedef boost::reverse_iterator< [y_iterator](#) > [reverse_y_iterator](#)
iterates backwards over v_{ij} with i fixed.

Public Member Functions

- [LexicographicalView](#) (const RandomAccessIterator &begin, const RandomAccessIterator &end, [Size](#) xSize)
attaches the view with the given dimension to a sequence

Element access

- [y_iterator](#) operator[] ([Size](#) i)

Iterator access

- [x_iterator](#) xbegin ([Size](#) j)
- [x_iterator](#) xend ([Size](#) j)
- [reverse_x_iterator](#) rxbegin ([Size](#) j)
- [reverse_x_iterator](#) rxend ([Size](#) j)
- [y_iterator](#) ybegin ([Size](#) i)
- [y_iterator](#) yend ([Size](#) i)
- [reverse_y_iterator](#) rybegin ([Size](#) i)
- [reverse_y_iterator](#) ryend ([Size](#) i)

Inspectors

- [Size xSize \(\)](#) const
dimension of the array along x
- [Size ySize \(\)](#) const
dimension of the array along y

7.305 Linear Class Reference

```
#include <ql/Math/interpolationtraits.hpp>
```

7.305.1 Detailed Description

Linear interpolation traits

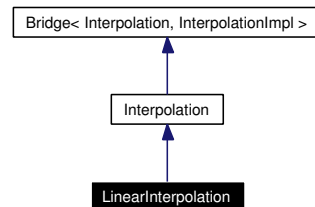
Static Public Member Functions

- `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)`

7.306 LinearInterpolation Class Reference

```
#include <ql/Math/linearinterpolation.hpp>
```

Inheritance diagram for LinearInterpolation:



7.306.1 Detailed Description

Linear interpolation between discrete points

Public Member Functions

- `template<class I1, class I2> LinearInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)`

7.306.2 Constructor & Destructor Documentation

7.306.2.1 [LinearInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

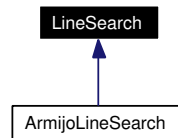
Precondition:

the *x* values must be sorted.

7.307 LineSearch Class Reference

```
#include <ql/Optimization/linesearch.hpp>
```

Inheritance diagram for LineSearch:



7.307.1 Detailed Description

Base class for line search.

Public Member Functions

- [LineSearch](#) ([Real](#) eps=1e-8)
Default constructor.
- virtual [~LineSearch](#) ()
Destructor.
- const [Array](#) & [lastX](#) ()
return last x value
- [Real](#) [lastFunctionValue](#) ()
return last cost function value
- const [Array](#) & [lastGradient](#) ()
return last gradient
- [Real](#) [lastGradientNorm2](#) ()
return square norm of last gradient
- bool [succeed](#) ()
- virtual [Real](#) [operator\(\)](#) (const [Problem](#) &P, [Real](#) t_ini)=0
Perform line search.
- [Real](#) [update](#) ([Array](#) ¶ms, const [Array](#) &direction, [Real](#) beta, const [Constraint](#) &constraint)

Protected Attributes

- [Array](#) [xtd_](#)
new x and its gradient
- [Array](#) [gradient_](#)

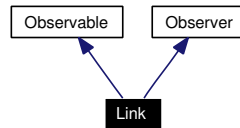
new x and its gradient

- [Real qt_](#)
cost function value and gradient norm corresponding to $xtd_$
- [Real qpt_](#)
cost function value and gradient norm corresponding to $xtd_$
- `bool` [succeed_](#)
flag to know if linesearch succeed

7.308 Link Class Template Reference

```
#include <ql/relinkablehandle.hpp>
```

Inheritance diagram for Link:



7.308.1 Detailed Description

```
template<class Type> class QuantLib::Link< Type >
```

Relinkable access to a shared pointer.

Precondition:

Class "Type" must inherit from [Observable](#)

Public Member Functions

- [Link](#) (const boost::shared_ptr< Type > &h=boost::shared_ptr< Type >(), bool registerAsObserver=true)
- void [linkTo](#) (const boost::shared_ptr< Type > &, bool registerAsObserver=true)
- bool [empty](#) () const
Checks if the contained shared pointer points to anything.
- bool [isNull](#) () const
Checks if the contained shared pointer points to anything.
- const boost::shared_ptr< Type > & [currentLink](#) () const
Returns the contained shared pointer.
- void [update](#) ()
[Observer](#) interface.

7.308.2 Constructor & Destructor Documentation

7.308.2.1 [Link](#) (const boost::shared_ptr< Type > &h = boost::shared_ptr< Type >(), bool registerAsObserver = true) [explicit]

Warning:

see the documentation of the [linkTo\(\)](#) method for issues relatives to registerAsObserver.

7.308.3 Member Function Documentation

7.308.3.1 `void linkTo (const boost::shared_ptr< Type > &, bool registerAsObserver = true)`

Warning:

`registerAsObserver` is left as a backdoor in case the programmer cannot guarantee that the object pointed to will remain alive for the whole lifetime of the handle—namely, it should be set to `false` when the passed shared pointer was created with `owns = false` (the latter should only happen in a controlled environment, so that the programmer is aware of it). Failure to do so can very likely result in a program crash. If the programmer does want the handle to register as observer of such a shared pointer, it is his responsibility to ensure that the handle gets destroyed before the pointed object does.

7.308.3.2 `bool isNull () const`

Checks if the contained shared pointer points to anything.

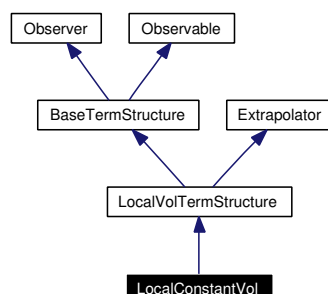
Deprecated

use `empty()` instead

7.309 LocalConstantVol Class Reference

```
#include <ql/Volatilities/localconstantvol.hpp>
```

Inheritance diagram for LocalConstantVol:



7.309.1 Detailed Description

Constant local volatility, no time-strike dependence.

This class implements the LocalVolatilityTermStructure interface for a constant local volatility (no time/asset dependence). Local volatility and Black volatility are the same when volatility is at most time dependent, so this class is basically a proxy for [BlackVolatilityTermStructure](#).

Public Member Functions

- **LocalConstantVol** (const [Date](#) &referenceDate, [Volatility](#) volatility, const [DayCounter](#) &dayCounter)
- **LocalConstantVol** (const [Date](#) &referenceDate, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)
- **LocalConstantVol** ([Integer](#) settlementDays, const [Calendar](#) &, [Volatility](#) volatility, const [DayCounter](#) &dayCounter)
- **LocalConstantVol** ([Integer](#) settlementDays, const [Calendar](#) &, const [Handle](#)< [Quote](#) > &volatility, const [DayCounter](#) &dayCounter)

LocalVolTermStructure interface

- [DayCounter](#) dayCounter () const
the day counter used for date/time conversion
- [Date](#) maxDate () const
the latest date for which the term structure can return vols
- [Real](#) minStrike () const
the minimum strike for which the term structure can return vols
- [Real](#) maxStrike () const
the maximum strike for which the term structure can return vols

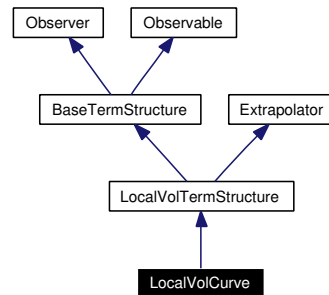
Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

7.310 LocalVolCurve Class Reference

```
#include <ql/Volatilities/localvolcurve.hpp>
```

Inheritance diagram for LocalVolCurve:



7.310.1 Detailed Description

Local volatility curve derived from a Black curve.

Public Member Functions

- **LocalVolCurve** (const [Handle](#)< [BlackVarianceCurve](#) > &curve)

LocalVolTermStructure interface

- const [Date](#) & [referenceDate](#) () const
the reference date, i.e., the date at which discount = 1
- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Date](#) [maxDate](#) () const
the latest date for which the term structure can return vols
- [Real](#) [minStrike](#) () const
the minimum strike for which the term structure can return vols
- [Real](#) [maxStrike](#) () const
the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

Protected Member Functions

- [Volatility](#) [localVolImpl](#) ([Time](#), [Real](#)) const

7.310.2 Member Function Documentation

7.310.2.1 [Volatility](#) localVolImpl ([Time](#) t , [Real](#) *dummy*) const [protected, virtual]

The relation

$$\int_0^T \sigma_L^2(t) dt = \sigma_B^2 T$$

holds, where $\sigma_L(t)$ is the local volatility at time t and $\sigma_B(T)$ is the Black volatility for maturity T . From the above, the formula

$$\sigma_L(t) = \sqrt{\frac{d}{dt} \sigma_B^2(t) t}$$

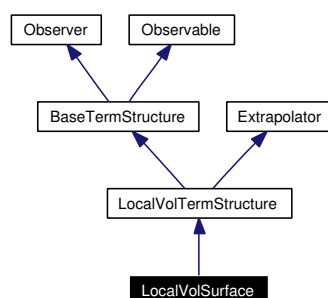
can be deduced which is here implemented.

Implements [LocalVolTermStructure](#).

7.311 LocalVolSurface Class Reference

```
#include <ql/Volatilities/localvolsurface.hpp>
```

Inheritance diagram for LocalVolSurface:



7.311.1 Detailed Description

Local volatility surface derived from a Black vol surface.

For details about this implementation refer to "Stochastic Volatility and Local Volatility," in "Case Studies and Financial Modelling Course Notes," by Jim Gatheral, Fall Term, 2003

see www.math.nyu.edu/fellows_fin_math/gatheral/Lecture1_Fall02.pdf

Bug

this class is untested, probably unreliable.

Public Member Functions

- **LocalVolSurface** (const [Handle](#)< [BlackVolTermStructure](#) > &blackTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, const [Handle](#)< [Quote](#) > &underlying)
- **LocalVolSurface** (const [Handle](#)< [BlackVolTermStructure](#) > &blackTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, [Real](#) underlying)

LocalVolTermStructure interface

- const [Date](#) & [referenceDate](#) () const
the reference date, i.e., the date at which discount = 1
- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Date](#) [maxDate](#) () const
the latest date for which the term structure can return vols
- [Real](#) [minStrike](#) () const
the minimum strike for which the term structure can return vols

- [Real maxStrike](#) () const
the maximum strike for which the term structure can return vols

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

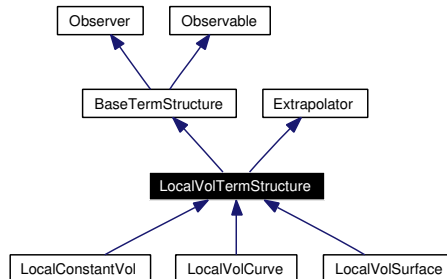
Protected Member Functions

- [Volatility localVolImpl](#) ([Time](#), [Real](#)) const
local vol calculation

7.312 LocalVolTermStructure Class Reference

```
#include <ql/voltermstructure.hpp>
```

Inheritance diagram for LocalVolTermStructure:



7.312.1 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

- virtual [DayCounter](#) [dayCounter](#) () const =0
the day counter used for date/time conversion

Constructors

See the [BaseTermStructure](#) documentation for issues regarding constructors.

- [LocalVolTermStructure](#) ()
default constructor
- [LocalVolTermStructure](#) (const [Date](#) &today, const [Date](#) &referenceDate)
initialize with a fixed reference date
- [LocalVolTermStructure](#) (const [Date](#) &referenceDate)
initialize with a fixed reference date
- [LocalVolTermStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)
calculate the reference date based on the global evaluation date

Local Volatility

- [Volatility](#) [localVol](#) (const [Date](#) &d, [Real](#) underlyingLevel, bool extrapolate=false) const
- [Volatility](#) [localVol](#) ([Time](#) t, [Real](#) underlyingLevel, bool extrapolate=false) const

Limits

- virtual [Date](#) `maxDate` () const =0
the latest date for which the term structure can return vols
- [Time](#) `maxTime` () const
the latest time for which the term structure can return vols
- virtual [Real](#) `minStrike` () const =0
the minimum strike for which the term structure can return vols
- virtual [Real](#) `maxStrike` () const =0
the maximum strike for which the term structure can return vols

Visitability

- virtual void `accept` ([AcyclicVisitor](#) &)

Protected Member Functions

Calculations

These methods must be implemented in derived classes to perform the actual volatility calculations. When they are called, range check has already been performed; therefore, they must assume that extrapolation is required.

- virtual [Volatility](#) `localVolImpl` ([Time](#) t, [Real](#) strike) const =0
local vol calculation

7.312.2 Constructor & Destructor Documentation

7.312.2.1 [LocalVolTermStructure](#) ()

default constructor

Warning:

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

7.313 LogLinear Class Reference

```
#include <ql/Math/interpolationtraits.hpp>
```

7.313.1 Detailed Description

Log-linear interpolation traits.

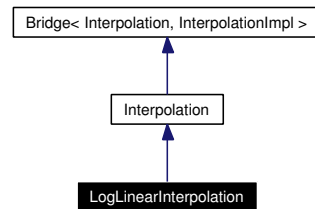
Static Public Member Functions

- `template<class I1, class I2> Interpolation make_interpolation` (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

7.314 LogLinearInterpolation Class Reference

```
#include <ql/Math/loglinearinterpolation.hpp>
```

Inheritance diagram for LogLinearInterpolation:



7.314.1 Detailed Description

Log-linear interpolation between discrete points

Todo

implement primitive, derivative, and secondDerivative functions.

Public Member Functions

- `template<class I1, class I2> LogLinearInterpolation (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)`

7.314.2 Constructor & Destructor Documentation

7.314.2.1 [LogLinearInterpolation](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

Precondition:

the *x* values must be sorted.

7.315 `lowest_category_iterator` Struct Template Reference

```
#include <ql/Utilities/iteratorcategories.hpp>
```

7.315.1 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](#)<std::random_access_iterator_tag, std::forward_iterator_tag>::iterator_category corresponds to std::forward_iterator_tag.

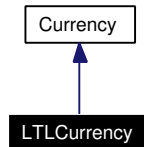
Deprecated

no longer needed after deprecation of [coupling_iterator](#)

7.316 LTLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for LTLCurrency:



7.316.1 Detailed Description

Lithuanian litas.

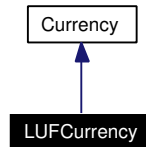
The ISO three-letter code is LTL; the numeric code is 440. It is divided in 100 centu.

ingroup currencies

7.317 LUFCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for LUFCurrency:



7.317.1 Detailed Description

Luxembourg franc.

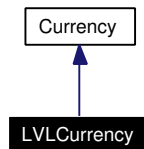
The ISO three-letter code is LUF; the numeric code is 442. It is divided in 100 centimes.

ingroup currencies

7.318 LVLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for LVLCurrency:



7.318.1 Detailed Description

Latvian lat.

The ISO three-letter code is LVL; the numeric code is 428. It is divided in 100 santims.

ingroup currencies

7.319 MakeSchedule Class Reference

```
#include <ql/schedule.hpp>
```

7.319.1 Detailed Description

helper class

This class provides a more comfortable interface to the argument list of Schedule's constructor.

Public Member Functions

- **MakeSchedule** (const [Calendar](#) &calendar, const [Date](#) &startDate, const [Date](#) &endDate, [Frequency](#) frequency, [BusinessDayConvention](#) convention)
- **MakeSchedule** & **withStubDate** (const [Date](#) &d)
- **MakeSchedule** & **backwards** (bool flag=true)
- **MakeSchedule** & **forwards** (bool flag=true)
- **MakeSchedule** & **longFinalPeriod** (bool flag=true)
- **MakeSchedule** & **shortFinalPeriod** (bool flag=true)
- **operator Schedule** ()

7.320 Matrix Class Reference

```
#include <ql/Math/matrix.hpp>
```

7.320.1 Detailed Description

Matrix used in linear algebra.

This class implements the concept of [Matrix](#) as used in linear algebra. As such, it is **not** meant to be used as a container.

Public Types

- typedef [Real](#) * **iterator**
- typedef const [Real](#) * **const_iterator**
- typedef boost::reverse_iterator< iterator > **reverse_iterator**
- typedef boost::reverse_iterator< const_iterator > **const_reverse_iterator**
- typedef [Real](#) * **row_iterator**
- typedef const [Real](#) * **const_row_iterator**
- typedef boost::reverse_iterator< row_iterator > **reverse_row_iterator**
- typedef boost::reverse_iterator< const_row_iterator > **const_reverse_row_iterator**
- typedef [step_iterator](#)< [Real](#) * > **column_iterator**
- typedef [step_iterator](#)< const [Real](#) * > **const_column_iterator**
- typedef boost::reverse_iterator< [column_iterator](#) > **reverse_column_iterator**
- typedef boost::reverse_iterator< [const_column_iterator](#) > **const_reverse_column_iterator**

Public Member Functions

Constructors, destructor, and assignment

- [Matrix](#) ()
creates a null matrix
- [Matrix](#) (Size rows, Size columns)
creates a matrix with the given dimensions
- [Matrix](#) (Size rows, Size columns, [Real](#) value)
creates the matrix and fills it with value
- [Matrix](#) (const [Matrix](#) &)
- [Matrix](#) (const [Disposable](#)< [Matrix](#) > &)
- [Matrix](#) & **operator=** (const [Matrix](#) &)
- [Matrix](#) & **operator=** (const [Disposable](#)< [Matrix](#) > &)

Algebraic operators

- const [Matrix](#) & **operator+=** (const [Matrix](#) &)
- const [Matrix](#) & **operator-=** (const [Matrix](#) &)
- const [Matrix](#) & **operator *=** ([Real](#))
- const [Matrix](#) & **operator/=** ([Real](#))

Iterator access

- `const_iterator` **begin** () const
- `iterator` **begin** ()
- `const_iterator` **end** () const
- `iterator` **end** ()
- `const_reverse_iterator` **rbegin** () const
- `reverse_iterator` **rbegin** ()
- `const_reverse_iterator` **rend** () const
- `reverse_iterator` **rend** ()
- `const_row_iterator` **row_begin** (Size i) const
- `row_iterator` **row_begin** (Size i)
- `const_row_iterator` **row_end** (Size i) const
- `row_iterator` **row_end** (Size i)
- `const_reverse_row_iterator` **row_rbegin** (Size i) const
- `reverse_row_iterator` **row_rbegin** (Size i)
- `const_reverse_row_iterator` **row_rend** (Size i) const
- `reverse_row_iterator` **row_rend** (Size i)
- `const_column_iterator` **column_begin** (Size i) const
- `column_iterator` **column_begin** (Size i)
- `const_column_iterator` **column_end** (Size i) const
- `column_iterator` **column_end** (Size i)
- `const_reverse_column_iterator` **column_rbegin** (Size i) const
- `reverse_column_iterator` **column_rbegin** (Size i)
- `const_reverse_column_iterator` **column_rend** (Size i) const
- `reverse_column_iterator` **column_rend** (Size i)

Element access

- `const_row_iterator` **operator[]** (Size) const
- `row_iterator` **operator[]** (Size)
- `Disposable< Array >` **diagonal** (void) const

Inspectors

- `Size` **rows** () const
- `Size` **columns** () const

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 &, Real)
- `const Disposable< Matrix >` **operator *** (Real, const Matrix &)
- `const Disposable< Matrix >` **operator/** (const Matrix &, Real)
- `const Disposable< Array >` **operator *** (const Array &, const Matrix &)

- `const Disposable< Array > operator * (const Matrix &, const Array &)`
- `const Disposable< Matrix > operator * (const Matrix &, const Matrix &)`
- `const Disposable< Matrix > transpose (const Matrix &)`
- `const Disposable< Matrix > outerProduct (const Array &v1, const Array &v2)`
- `template<class Iterator1, class Iterator2> const Disposable< Matrix > outerProduct (Iterator1 v1begin, Iterator1 v1end, Iterator2 v2begin, Iterator2 v2end)`
- `const Disposable< Matrix > pseudoSqrt (const Matrix &, SalvagingAlgorithm::Type)`

Returns the pseudo square root of a real symmetric matrix.

- `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real componentRetainedPercentage, SalvagingAlgorithm::Type)`

7.320.2 Member Function Documentation

7.320.2.1 `const Matrix & operator+= (const Matrix &)`

Precondition:

all matrices involved in an algebraic expression must have the same size.

7.320.3 Friends And Related Function Documentation

7.320.3.1 `std::ostream & operator<< (std::ostream &, const Matrix &)` [related]

Deprecated

send to the stream the output of `MatrixFormatter`

7.320.3.2 `const Disposable< Matrix > pseudoSqrt (const Matrix &, SalvagingAlgorithm::Type)` [related]

Returns the pseudo square root of a real symmetric matrix.

Given a matrix M , the result S is defined as the matrix such that $SS^T = M$. If the matrix is not positive semi definite, it can return an approximation of the pseudo square root using a (user selected) salvaging algorithm.

For more information see: "The most general methodology to create a valid correlation matrix for risk management and option pricing purposes", by R. Rebonato and P. Jäckel. The Journal of Risk, 2(2), Winter 1999/2000 <http://www.rebonato.com/correlationmatrix.pdf>

Revised and extended in "Monte Carlo Methods in Finance", by Peter Jäckel, Chapter 6.

Precondition:

the given matrix must be symmetric.

Todo

implement Hypersphere decomposition: 1) Jäckel "Monte Carlo Methods in Finance", Chapter 6 2) Brigo "A Note on Correlation and Rank Reduction" 3) Rapisarda, Brigo, Mercurio "Parameterizing correlations: a geometric interpretation"
implement Higham algorithm: Higham "Computing the nearest correlation matrix"

Tests

- a) the correctness of the results is tested by reproducing known good data.
- b) the correctness of the results is tested by checking returned values against numerical calculations.

7.320.3.3 `const Disposable< Matrix > rankReducedSqrt (const Matrix &, Size maxRank, Real componentRetainedPercentage, SalvagingAlgorithm::Type)` [related]

Precondition:

the given matrix must be symmetric.

7.321 MatrixFormatter Class Reference

```
#include <ql/Math/matrix.hpp>
```

7.321.1 Detailed Description

format matrices for output

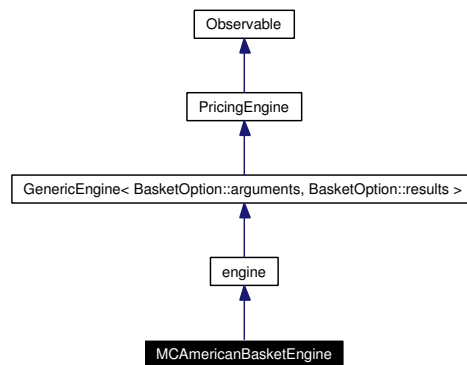
Static Public Member Functions

- `std::string toString` (const [Matrix](#) &m, [Integer](#) precision=6, [Integer](#) digits=0)

7.322 MCAmericanBasketEngine Class Reference

```
#include <ql/PricingEngines/Basket/mcamericanbasketengine.hpp>
```

Inheritance diagram for MCAmericanBasketEngine:



7.322.1 Detailed Description

least-square Monte Carlo engine

Warning:

This method is intrinsically weak for out-of-the-money options.

Bug

this engine does not yet work for put options. More problems might surface.

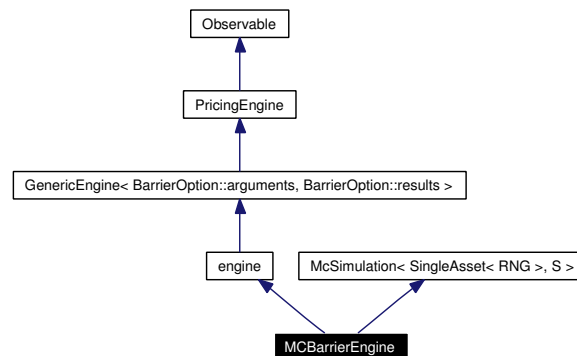
Public Member Functions

- **MCAmericanBasketEngine** ([Size](#) requiredSamples, [Size](#) timeSteps, [BigNatural](#) seed=0)
- void **calculate** () const

7.323 MCBarrierEngine Class Template Reference

```
#include <ql/PricingEngines/Barrier/mcbarrierengine.hpp>
```

Inheritance diagram for MCBarrierEngine:



7.323.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCBarrierEngine< RNG, S >
```

Pricing engine for barrier options using Monte Carlo simulation.

Uses the Brownian-bridge correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic Option Valuation* - D.R. Beaglehole, P.H. Dybvig and G. Zhou *Financial Analysts Journal*; Jan/Feb 1997; 53, 1. pg. 62-68 and *Simulating path-dependent options: A new approach* - M. El Babsiri and G. Noel *Journal of Derivatives*; Winter 1998; 6, 2; pg. 65-83

Tests

the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef [McSimulation](#)< 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**

Public Member Functions

- **MCBarrierEngine** ([Size](#) maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, bool isBiased, [BigNatural](#) seed)
- void **calculate** () const

Protected Member Functions

- [TimeGrid](#) **timeGrid** () const

- `boost::shared_ptr< path_generator_type > pathGenerator () const`
- `boost::shared_ptr< path_pricer_type > pathPricer () const`

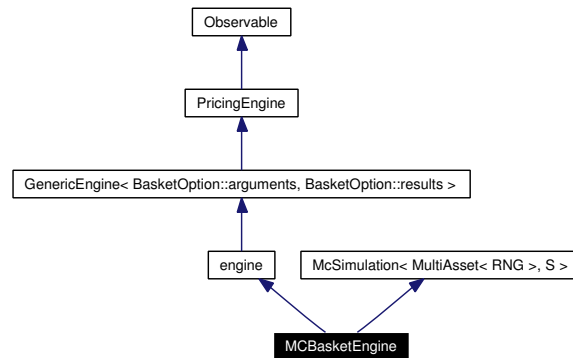
Protected Attributes

- `Size maxTimeStepsPerYear_`
- `Size requiredSamples_`
- `Size maxSamples_`
- `Real requiredTolerance_`
- `bool isBiased_`
- `bool brownianBridge_`
- `BigNatural seed_`

7.324 MCBasketEngine Class Template Reference

```
#include <ql/PricingEngines/Basket/mcbasketengine.hpp>
```

Inheritance diagram for MCBasketEngine:



7.324.1 Detailed Description

`template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCBasketEngine< RNG, S >`

Pricing engine for basket options using Monte Carlo simulation.

Tests

the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef `McSimulation< 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` brownianBridge, `bool` antitheticVariate, `bool` controlVariate, `Size` requiredSamples, `Real` requiredTolerance, `Size` maxSamples, `BigNatural` seed)
- `void calculate () const`

Protected Member Functions

- `TimeGrid timeGrid () const`
- `boost::shared_ptr< path_generator_type > pathGenerator () const`
- `boost::shared_ptr< path_pricer_type > pathPricer () const`

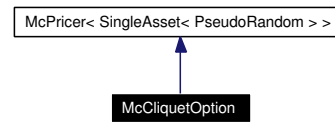
Protected Attributes

- [Size](#) `maxTimeStepsPerYear_`
- [Size](#) `requiredSamples_`
- [Size](#) `maxSamples_`
- [Real](#) `requiredTolerance_`
- `bool brownianBridge_`
- `BigNatural seed_`

7.325 McCliquetOption Class Reference

```
#include <ql/Pricers/mccliquetoption.hpp>
```

Inheritance diagram for McCliquetOption:



7.325.1 Detailed Description

simple example of Monte Carlo pricer

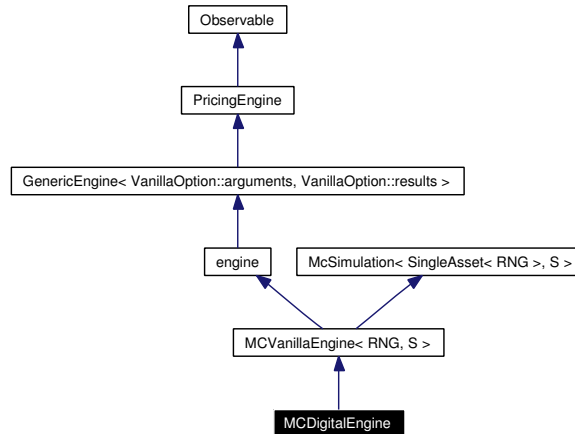
Public Member Functions

- **McCliquetOption** (Option::Type type, [Real](#) underlying, [Real](#) moneyiness, const [Handle](#)<[YieldTermStructure](#) > ÷ndYield, const [Handle](#)<[YieldTermStructure](#) > &riskFreeRate, const [Handle](#)<[BlackVolTermStructure](#) > &volatility, const std::vector<[Time](#) > ×, [Real](#) accruedCoupon, [Real](#) lastFixing, [Real](#) localCap, [Real](#) localFloor, [Real](#) globalCap, [Real](#) globalFloor, bool redemptionOnly, [BigNatural](#) seed=0)

7.326 MCDigitalEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mcdigitalengine.hpp>
```

Inheritance diagram for MCDigitalEngine:



7.326.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDigital-
Engine< RNG, S >
```

Pricing engine for digital options using Monte Carlo simulation.

Uses the Brownian [Bridge](#) correction for the barrier found in *Going to Extremes: Correcting Simulation Bias in Exotic [Option](#) Valuation* - D.R. Beaglehole, P.H. Dybvig and G. Zhou *Financial Analysts Journal*; Jan/Feb 1997; 53, 1. pg. 62-68 and *Simulating path-dependent options: A new approach* - M. El Babsiri and G. Noel *Journal of Derivatives*; Winter 1998; 6, 2; pg. 65-83

Tests

the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing known good results.

Public Types

- typedef [MCVanillaEngine](#)< RNG, S >::path_generator_type **path_generator_type**
- typedef [MCVanillaEngine](#)< RNG, S >::path_pricer_type **path_pricer_type**
- typedef [MCVanillaEngine](#)< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCDigitalEngine** ([Size](#) maxTimeStepsPerYear, bool brownianBridge, bool antithetic-Variate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) max-Samples, BigNatural seed)

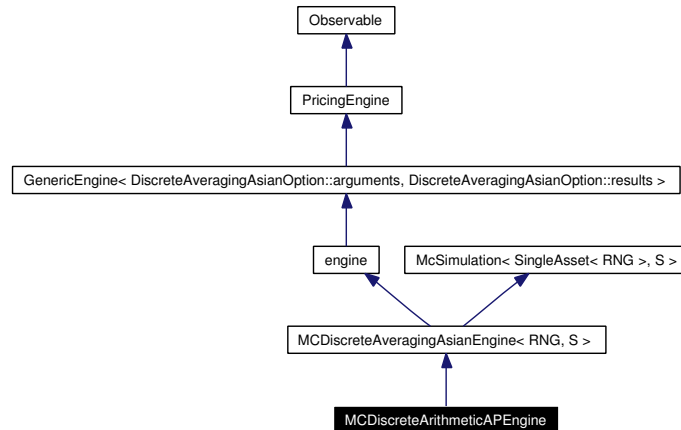
Protected Member Functions

- [TimeGrid](#) **timeGrid** () const
- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.327 MCDiscreteArithmeticAPEngine Class Template Reference

```
#include <ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp>
```

Inheritance diagram for MCDiscreteArithmeticAPEngine:



7.327.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscreteArithmeticAPEngine< RNG, S >

Monte Carlo pricing engine for discrete arithmetic average price Asian.

Monte Carlo pricing engine for discrete arithmetic average price Asian options. It can use [MCDiscreteGeometricAPEngine](#) (Monte Carlo discrete arithmetic average price engine) and [AnalyticDiscreteGeometricAveragePriceAsianEngine](#) (analytic discrete arithmetic average price engine) for control variation.

Tests

the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::path_generator_type **path_generator_type**
- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::path_pricer_type **path_pricer_type**
- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCDiscreteArithmeticAPEngine** ([Size](#) maxTimeStepPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, [BigNatural](#))

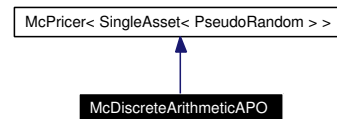
Protected Member Functions

- `boost::shared_ptr< path_pricer_type > pathPricer () const`
- `boost::shared_ptr< path_pricer_type > controlPathPricer () const`
- `boost::shared_ptr< PricingEngine > controlPricingEngine () const`

7.328 McDiscreteArithmeticAPO Class Reference

```
#include <ql/Pricers/mcdiscretearithmeticapo.hpp>
```

Inheritance diagram for McDiscreteArithmeticAPO:



7.328.1 Detailed Description

example of Monte Carlo pricer using a control variate

Deprecated

use the [DiscreteAveragingAsianOption](#) instrument with [MCDiscreteArithmeticAPEngine](#) instead

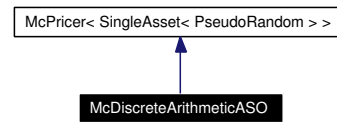
Public Member Functions

- **McDiscreteArithmeticAPO** (Option::Type type, [Real](#) underlying, [Real](#) strike, const [Handle](#)< [YieldTermStructure](#) > ÷ndYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > ×, bool controlVariate, BigNatural seed=0)

7.329 McDiscreteArithmeticASO Class Reference

```
#include <ql/Pricers/mcdiscretearithmeticaso.hpp>
```

Inheritance diagram for McDiscreteArithmeticASO:



7.329.1 Detailed Description

example of Monte Carlo pricer using a control variate.

Todo

continous-averaging version

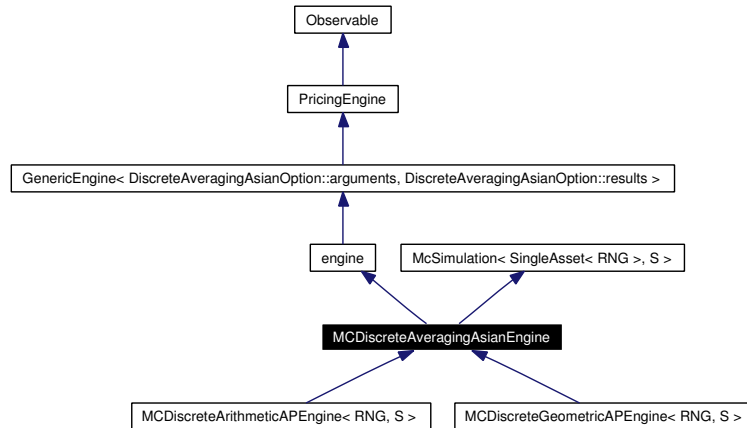
Public Member Functions

- **McDiscreteArithmeticASO** (Option::Type type, [Real](#) underlying, const [Handle](#)< [YieldTermStructure](#) > ÷ndYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > ×, bool controlVariate, BigNatural seed=0)

7.330 MCDiscreteAveragingAsianEngine Class Template Reference

```
#include <ql/PricingEngines/Asian/mcdiscreteasianengine.hpp>
```

Inheritance diagram for MCDiscreteAveragingAsianEngine:



7.330.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscreteAveragingAsianEngine< RNG, S >
```

Pricing engine for discrete average Asians using Monte Carlo simulation.

Warning:

control-variate calculation is disabled under VC++6

Public Types

- typedef [McSimulation](#)< 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**

Public Member Functions

- **MCDiscreteAveragingAsianEngine** ([Size](#) maxTimeStepsPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, BigNatural seed)
- void **calculate** () const

Protected Member Functions

- [TimeGrid](#) **timeGrid** () const

- `boost::shared_ptr< path_generator_type > pathGenerator () const`
- `Real controlVariateValue () const`

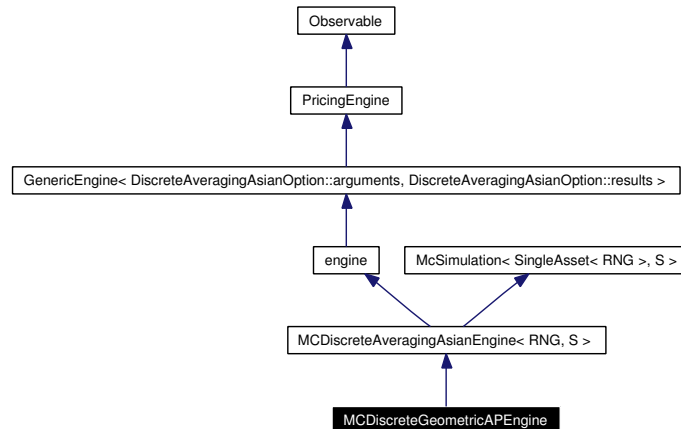
Protected Attributes

- `Size maxTimeStepsPerYear_`
- `Size requiredSamples_`
- `Size maxSamples_`
- `Real requiredTolerance_`
- `bool brownianBridge_`
- `BigNatural seed_`

7.331 MCDiscreteGeometricAPEngine Class Template Reference

```
#include <ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp>
```

Inheritance diagram for MCDiscreteGeometricAPEngine:



7.331.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCDiscreteGeometricAPEngine< RNG, S >
```

Monte Carlo pricing engine for discrete geometric average price Asian.

Tests

the correctness of the returned value is tested by reproducing results available in literature.

Public Types

- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::path_generator_type **path_generator_type**
- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::path_pricer_type **path_pricer_type**
- typedef [MCDiscreteAveragingAsianEngine](#)< RNG, S >::stats_type **stats_type**

Public Member Functions

- **MCDiscreteGeometricAPEngine** ([Size](#) maxTimeStepPerYear, bool brownianBridge, bool antitheticVariate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, BigNatural seed)

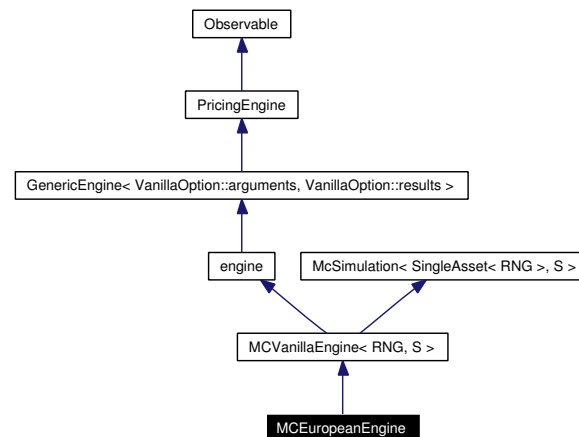
Protected Member Functions

- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.332 MCEuropeanEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mceuropeanengine.hpp>
```

Inheritance diagram for MCEuropeanEngine:



7.332.1 Detailed Description

template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCEuropeanEngine< RNG, S >

European option pricing engine using Monte Carlo simulation.

Tests

the correctness of the returned value is tested by checking it against analytic results.

Public Types

- typedef [MCVanillaEngine](#)< 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 brownianBridge, bool antitheticVariate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, BigNatural seed)

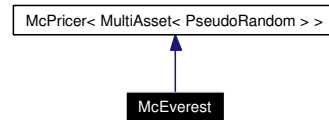
Protected Member Functions

- [TimeGrid](#) **timeGrid** () const
- boost::shared_ptr< path_pricer_type > **pathPricer** () const

7.333 McEverest Class Reference

```
#include <ql/Pricers/mceverest.hpp>
```

Inheritance diagram for McEverest:



7.333.1 Detailed Description

Everest-type option pricer.

The payoff of an Everest option is simply given by the final price / initial price ratio of the worst performer

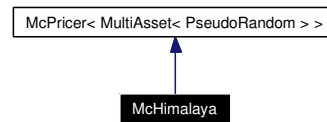
Public Member Functions

- **McEverest** (const std::vector< [Handle](#)< [YieldTermStructure](#) > > ÷ndYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const std::vector< [Handle](#)< [BlackVolTermStructure](#) > > &volatilities, const [Matrix](#) &correlation, [Time](#) residualTime, [BigNatural](#) seed=0)

7.334 McHimalaya Class Reference

```
#include <ql/Pricers/mchimalaya.hpp>
```

Inheritance diagram for McHimalaya:



7.334.1 Detailed Description

Himalayan-type option pricer.

The payoff of a Himalaya option is computed in the following way: Given a basket of N assets, and N time periods, at end of each period the option who performed the best is added to the average and then discarded from the basket. At the end of the N periods the option pays the max between the strike and the average of the best performers.

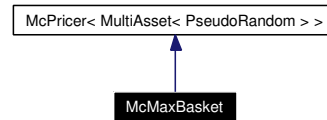
Public Member Functions

- **McHimalaya** (const std::vector< [Real](#) > &underlyings, const std::vector< [Handle](#)< [YieldTermStructure](#) > > ÷ndYields, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const std::vector< [Handle](#)< [BlackVolTermStructure](#) > > &volatilities, const [Matrix](#) &correlation, [Real](#) strike, const std::vector< [Time](#) > ×, [BigNatural](#) seed=0)

7.335 McMaxBasket Class Reference

```
#include <ql/Pricers/mcmaxbasket.hpp>
```

Inheritance diagram for McMaxBasket:



7.335.1 Detailed Description

simple example of multi-factor Monte Carlo pricer

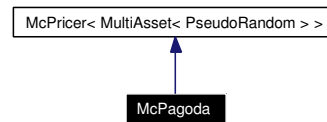
Public Member Functions

- **McMaxBasket** (const std::vector< [Real](#) > &underlyings, const std::vector< [Handle](#)< [YieldTermStructure](#) > > ÷ndYields, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const std::vector< [Handle](#)< [BlackVolTermStructure](#) > > &volatilities, const [Matrix](#) &correlation, [Time](#) residualTime, [BigNatural](#) seed=0)

7.336 McPagoda Class Reference

```
#include <ql/Pricers/mcpagoda.hpp>
```

Inheritance diagram for McPagoda:



7.336.1 Detailed Description

roofed Asian option

Given a certain portfolio of assets at the end of the period it is returned the minimum of a given roof and a certain fraction of the positive portfolio performance. If the performance of the portfolio is below then the payoff is null.

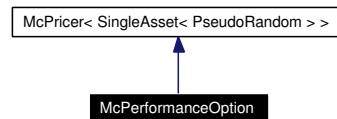
Public Member Functions

- **McPagoda** (const std::vector< Real > &underlyings, Real fraction, Real roof, const std::vector< Handle< YieldTermStructure > > ÷ndYields, const Handle< YieldTermStructure > &riskFreeRate, const std::vector< Handle< BlackVolTermStructure > > &volatilities, const Matrix &correlation, const std::vector< Time > ×, BigNatural seed=0)

7.337 McPerformanceOption Class Reference

```
#include <ql/Pricers/mcperformanceoption.hpp>
```

Inheritance diagram for McPerformanceOption:



7.337.1 Detailed Description

Performance option computed using Monte Carlo simulation.

A performance option is a variant of a cliquet option: the payoff of each forward-starting (a.k.a. deferred strike) options is \$ $\max(S/X - 1)$ \$.

Public Member Functions

- **McPerformanceOption** (Option::Type type, [Real](#) underlying, [Real](#) moneyness, const [Handle](#)< [YieldTermStructure](#) > ÷ndYield, const [Handle](#)< [YieldTermStructure](#) > &riskFreeRate, const [Handle](#)< [BlackVolTermStructure](#) > &volatility, const std::vector< [Time](#) > ×, [BigNatural](#) seed=0)

7.338 McPricer Class Template Reference

```
#include <ql/Pricers/mcpricer.hpp>
```

7.338.1 Detailed Description

```
template<class MC, class S = Statistics> class QuantLib::McPricer< MC, S >
```

base class for Monte Carlo pricers

Eventually this class might be linked to the general tree of pricers, in order to have tools like `impliedVolatility` available. Also, it could, eventually, offer greeks methods. Deriving a class from `McPricer` gives an easy way to write a Monte Carlo Pricer. See `McEuropean` as example of one factor pricer, `Basket` as example of multi factor pricer.

Public Member Functions

- `Real value` (`Real` tolerance, `Size` maxSample=QL_MAX_INTEGER) const
add samples until the required tolerance is reached
- `Real valueWithSamples` (`Size` samples) const
simulate a fixed number of samples
- `Real errorEstimate` () const
error Estimated of the samples simulated so far
- `const S & sampleAccumulator` (void) const
access to the sample accumulator for more statistics

Protected Attributes

- `boost::shared_ptr< MonteCarloModel< MC, S > > mcModel_`

Static Protected Attributes

- `const Size minSample_ = 1023`

7.339 McSimulation Class Template Reference

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

7.339.1 Detailed Description

```
template<class MC, class S = Statistics> class QuantLib::McSimulation< MC, S >
```

base class for Monte Carlo engines

Eventually this class might offer greeks methods. Deriving a class from McEngine gives an easy way to write a Monte Carlo engine.

See McVanillaEngine as an example of one factor engine.

Public Types

- typedef [MonteCarloModel](#)< MC, S >::path_generator_type **path_generator_type**
- typedef [MonteCarloModel](#)< MC, S >::path_pricer_type **path_pricer_type**
- typedef [MonteCarloModel](#)< MC, S >::stats_type **stats_type**

Public Member Functions

- [Real](#) value ([Real](#) tolerance, [Size](#) maxSample=QL_MAX_INTEGER) const
add samples until the required absolute tolerance is reached
- [Real](#) valueWithSamples ([Size](#) samples) const
simulate a fixed number of samples
- [Real](#) errorEstimate () const
error estimated using the samples simulated so far
- const stats_type & [sampleAccumulator](#) (void) const
access to the sample accumulator for richer statistics
- void [calculate](#) ([Real](#) requiredTolerance, [Size](#) requiredSamples, [Size](#) maxSamples) const
basic calculate method provided to inherited pricing engines

Protected Member Functions

- **McSimulation** (bool antitheticVariate, bool controlVariate)
- virtual boost::shared_ptr< path_pricer_type > **pathPricer** () const =0
- virtual boost::shared_ptr< path_generator_type > **pathGenerator** () const =0
- virtual [TimeGrid](#) **timeGrid** () const =0
- virtual boost::shared_ptr< path_pricer_type > **controlPathPricer** () const
- virtual boost::shared_ptr< [PricingEngine](#) > **controlPricingEngine** () const
- virtual [Real](#) **controlVariateValue** () const

Protected Attributes

- boost::shared_ptr< [MonteCarloModel](#)< MC, S > > **mcModel_**
- bool **antitheticVariate_**
- bool **controlVariate_**

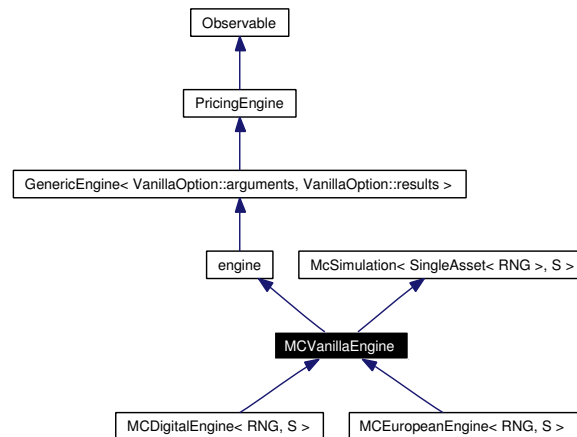
Static Protected Attributes

- const [Size](#) **minSample_** = 1023

7.340 MCVanillaEngine Class Template Reference

```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

Inheritance diagram for MCVanillaEngine:



7.340.1 Detailed Description

```
template<class RNG = PseudoRandom, class S = Statistics> class QuantLib::MCVanillaEngine< RNG, S >
```

Pricing engine for vanilla options using Monte Carlo simulation.

Public Member Functions

- void **calculate** () const

Protected Types

- typedef [McSimulation](#)< 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 brownianBridge, bool antitheticVariate, bool controlVariate, [Size](#) requiredSamples, [Real](#) requiredTolerance, [Size](#) maxSamples, BigNatural seed)
- boost::shared_ptr< path_generator_type > **pathGenerator** () const
- [Real](#) **controlVariateValue** () const

Protected Attributes

- [Size](#) maxTimeStepsPerYear_
- [Size](#) requiredSamples_
- [Size](#) maxSamples_
- [Real](#) requiredTolerance_
- bool brownianBridge_
- [BigNatural](#) seed_

7.341 MersenneTwisterUniformRng Class Reference

```
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

7.341.1 Detailed Description

Uniform random number generator.

Mersenne Twister random number generator of period $2^{19937}-1$

For more details see <http://www.math.keio.ac.jp/matsumoto/emt.html>

Tests

the correctness of the returned values is tested by checking them against known good results.

Public Types

- typedef [Sample](#)< [Real](#) > **sample_type**

Public Member Functions

- [MersenneTwisterUniformRng](#) (unsigned long seed=0)
- [MersenneTwisterUniformRng](#) (const std::vector< unsigned long > &seeds)
- [sample_type](#) next () const
- unsigned long [nextInt32](#) () const
return a random number on $[0, 0xffffffff]$ -interval

7.341.2 Constructor & Destructor Documentation

7.341.2.1 [MersenneTwisterUniformRng](#) (unsigned long seed = 0) [explicit]

if the given seed is 0, a random seed will be chosen based on clock()

7.341.3 Member Function Documentation

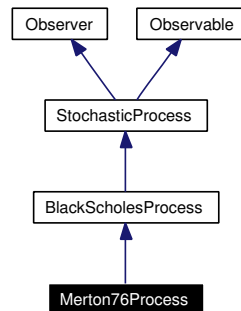
7.341.3.1 [sample_type](#) next () const

returns a sample with weight 1.0 containing a random number on (0.0, 1.0)-real-interval

7.342 Merton76Process Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for Merton76Process:



7.342.1 Detailed Description

Merton-76 jump-diffusion process.

Public Member Functions

- **Merton76Process** (const [Handle](#)< [Quote](#) > &stateVariable, const [Handle](#)< [YieldTermStructure](#) > ÷ndTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &blackVolTS, const [Handle](#)< [Quote](#) > &jumpInt, const [Handle](#)< [Quote](#) > &logJMean, const [Handle](#)< [Quote](#) > &logJVol, const boost::shared_ptr< [StochasticProcess::discretization](#) > &d=boost::shared_ptr< [StochasticProcess::discretization](#) >(new [EulerDiscretization](#)))

StochasticProcess interface

- [Real](#) drift ([Time](#), [Real](#)) const
- [Real](#) diffusion ([Time](#), [Real](#)) const
- [Real](#) evolve ([Real](#) change, [Real](#) currentValue) const

Inspectors

- const boost::shared_ptr< [Quote](#) > & **jumpIntensity** () const
- const boost::shared_ptr< [Quote](#) > & **logMeanJump** () const
- const boost::shared_ptr< [Quote](#) > & **logJumpVolatility** () const

7.342.2 Member Function Documentation

7.342.2.1 [Real](#) drift ([Time](#), [Real](#)) const [virtual]

Todo

revise extrapolation

Reimplemented from [BlackScholesProcess](#).

7.342.2.2 **Real** diffusion (**Time**, **Real**) const [virtual]

Todo

revise extrapolation

Reimplemented from [BlackScholesProcess](#).

7.342.2.3 **Real** evolve (**Real** *change*, **Real** *currentValue*) const [virtual]

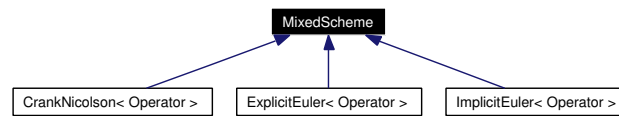
applies a change to the asset value. By default; it returns $x + \Delta x$.

Reimplemented from [BlackScholesProcess](#).

7.343 MixedScheme Class Template Reference

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Inheritance diagram for MixedScheme:



7.343.1 Detailed Description

```
template<class Operator> class QuantLib::MixedScheme< Operator >
```

Mixed (explicit/implicit) scheme for finite difference methods.

See sect. [Finite-differences framework](#) for details on the method.

In this implementation, the passed operator must be derived from either `TimeConstantOperator` or `TimeDependentOperator`. Also, it must implement at least the following interface:

```
typedef ... 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*(Real, const Operator&);
Operator operator+(const Operator&, const Operator&);
Operator operator+(const Operator&, const Operator&);
```

Warning:

The differential operator must be linear for this evolver to work.

Todo

derive variable theta schemes
introduce multi time-level schemes.

Protected Types

- typedef `Operator::arrayType` **arrayType**
- typedef `Operator` **operatorType**
- typedef `BoundaryCondition< Operator >` **bcType**

Protected Member Functions

- **MixedScheme** (const Operator &L, [Real](#) theta, const std::vector< boost::shared_ptr< [bcType](#) > > &bcs)
- void **step** (arrayType &a, [Time](#) t)
- void **setStep** ([Time](#) dt)

Protected Attributes

- Operator L_
- Operator I_
- Operator **explicitPart**_
- Operator **implicitPart**_
- [Time](#) dt_
- [Real](#) theta_
- std::vector< boost::shared_ptr< [bcType](#) > > bcs_

Friends

- class **FiniteDifferenceModel**<MixedScheme<Operator> >

7.344 Money Class Reference

```
#include <ql/money.hpp>
```

7.344.1 Detailed Description

amount of cash

Tests

money arithmetic is tested with and without currency conversions.

[NOHEADER]

- enum [ConversionType](#) { [NoConversion](#), [BaseCurrencyConversion](#), [AutomatedConversion](#) }
- [ConversionType](#) [conversionType](#)
- [Currency](#) [baseCurrency](#)

Public Member Functions

Constructors

- [Money](#) (const [Currency](#) ¤cy, [Decimal](#) value)
- [Money](#) ([Decimal](#) value, const [Currency](#) ¤cy)

Inspectors

- const [Currency](#) & [currency](#) () const
- [Decimal](#) [value](#) () const
- [Money](#) [rounded](#) () const

Money arithmetics

See below for non-member functions and for settings which determine the behavior of the operators.

- [Money](#) [operator+](#) () const
- [Money](#) [operator-](#) () const
- [Money](#) & [operator+=](#) (const [Money](#) &)
- [Money](#) & [operator-=](#) (const [Money](#) &)
- [Money](#) & [operator*=](#) ([Decimal](#))
- [Money](#) & [operator/=](#) ([Decimal](#))

Related Functions

(Note that these are not member functions.)

- [Money](#) [operator+](#) (const [Money](#) &, const [Money](#) &)
- [Money](#) [operator-](#) (const [Money](#) &, const [Money](#) &)
- [Money](#) [operator*](#) (const [Money](#) &, [Decimal](#))
- [Money](#) [operator*](#) ([Decimal](#), const [Money](#) &)
- [Money](#) [operator/](#) (const [Money](#) &, [Decimal](#))

- **Decimal operator/** (const [Money](#) &, const [Money](#) &)
- **bool operator==** (const [Money](#) &, const [Money](#) &)
- **bool operator!=** (const [Money](#) &, const [Money](#) &)
- **bool operator<** (const [Money](#) &, const [Money](#) &)
- **bool operator<=** (const [Money](#) &, const [Money](#) &)
- **bool operator>** (const [Money](#) &, const [Money](#) &)
- **bool operator>=** (const [Money](#) &, const [Money](#) &)
- **bool close** (const [Money](#) &, const [Money](#) &, [Size](#) n=42)
- **bool close_enough** (const [Money](#) &, const [Money](#) &, [Size](#) n=42)

7.344.2 Member Enumeration Documentation

7.344.2.1 enum [ConversionType](#)

Conversion settings

These parameters are used for combining money amounts in different currencies

Enumeration values:

NoConversion do not perform conversions

BaseCurrencyConversion convert both operands to the base currency before converting

AutomatedConversion return the result in the currency of the first operand

7.345 MoneyFormatter Class Reference

```
#include <ql/money.hpp>
```

7.345.1 Detailed Description

format money for output

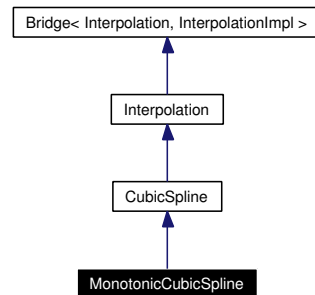
Static Public Member Functions

- `std::string toString (const Money &)`

7.346 MonotonicCubicSpline Class Reference

#include <ql/Math/cubicspline.hpp>

Inheritance diagram for MonotonicCubicSpline:



7.346.1 Detailed Description

Cubic spline with monotonicity constraint

Public Member Functions

- template<class I1, class I2> [MonotonicCubicSpline](#) (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin, [CubicSpline::BoundaryCondition](#) leftCondition, [Real](#) leftConditionValue, [CubicSpline::BoundaryCondition](#) rightCondition, [Real](#) rightConditionValue)

7.346.2 Constructor & Destructor Documentation

- 7.346.2.1 [MonotonicCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*, [CubicSpline::BoundaryCondition](#) *leftCondition*, [Real](#) *leftConditionValue*, [CubicSpline::BoundaryCondition](#) *rightCondition*, [Real](#) *rightConditionValue*)

Precondition:

the *x* values must be sorted.

7.347 MonteCarloModel Class Template Reference

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

7.347.1 Detailed Description

```
template<class mc_traits, class stats_traits = Statistics> class QuantLib::MonteCarloModel<
mc_traits, stats_traits >
```

General purpose Monte Carlo model for path samples.

The template arguments of this class correspond to available policies for the particular model to be instantiated—i.e., whether it is single- or multi-asset, or whether it should use pseudo-random or low-discrepancy numbers for path generation. Such decisions are grouped in trait classes so as to be orthogonal—see [mctrails.hpp](#) for examples.

The constructor accepts two safe references, i.e. two smart pointers, one to a path generator and the other to a path pricer. In case of control variate technique the user should provide the additional control option, namely the option path pricer and the option value.

Public Types

- typedef mc_traits::rsg_type **rsg_type**
- typedef mc_traits::path_generator_type **path_generator_type**
- typedef mc_traits::path_pricer_type **path_pricer_type**
- typedef path_generator_type::sample_type **sample_type**
- typedef path_pricer_type::result_type **result_type**
- typedef stats_traits **stats_type**

Public Member Functions

- **MonteCarloModel** (const boost::shared_ptr< path_generator_type > &pathGenerator, const boost::shared_ptr< path_pricer_type > &pathPricer, const stats_type &sampleAccumulator, bool antitheticVariate, const boost::shared_ptr< path_pricer_type > &cvPathPricer=boost::shared_ptr< path_pricer_type >(), result_type cvOptionValue=result_type())
- void **addSamples** ([Size](#) samples)
- const stats_type & **sampleAccumulator** (void) const

7.348 MoreGreeks Class Reference

```
#include <ql/option.hpp>
```

Inheritance diagram for MoreGreeks:



7.348.1 Detailed Description

more additional option results

Public Member Functions

- `void reset ()`

Public Attributes

- [Real](#) itmCashProbability
- [Real](#) deltaForward
- [Real](#) elasticity
- [Real](#) thetaPerDay
- [Real](#) strikeSensitivity

7.349 MoroInverseCumulativeNormal Class Reference

```
#include <ql/Math/normaldistribution.hpp>
```

7.349.1 Detailed Description

Moro Inverse cumulative normal distribution class.

Given x between zero and one as the integral value of a gaussian normal distribution this class provides the value y such that formula here ...

It uses Beasly and Springer approximation, with an improved approximation for the tails. See Boris Moro, "The Full Monte", 1995, Risk Magazine.

This class can also be used to generate a gaussian normal distribution from a uniform distribution. This is especially useful when a gaussian normal distribution is generated from a low discrepancy uniform distribution: in this case the traditional Box-Muller approach and its variants would not preserve the sequence's low-discrepancy.

Peter J. Acklam's approximation is better and is available as [QuantLib::InverseCumulativeNormal](#)

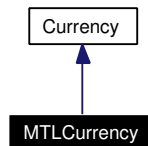
Public Member Functions

- **MoroInverseCumulativeNormal** ([Real](#) average=0.0, [Real](#) sigma=1.0)
- **Real operator()** ([Real](#) x) const

7.350 MTLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for MTLCurrency:



7.350.1 Detailed Description

Maltese lira.

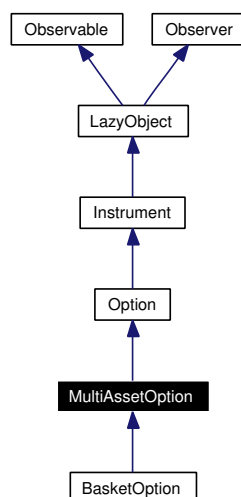
The ISO three-letter code is MTL; the numeric code is 470. It is divided in 100 cents.

ingroup currencies

7.351 MultiAssetOption Class Reference

```
#include <ql/Instruments/multiassetoption.hpp>
```

Inheritance diagram for MultiAssetOption:



7.351.1 Detailed Description

Base class for options on multiple assets.

Public Member Functions

- **MultiAssetOption** (const std::vector< boost::shared_ptr< [BlackScholesProcess](#) > &stoch-
Procs, const boost::shared_ptr< [Payoff](#) > &payoff, const boost::shared_ptr< [Exercise](#)
> &exercise, const [Matrix](#) &correlation, const boost::shared_ptr< [PricingEngine](#) >
&engine=boost::shared_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) *) const

Instrument interface

- bool [isExpired](#) () const
returns whether the instrument is still tradable.

greeks

- [Real](#) [delta](#) () const
- [Real](#) [gamma](#) () const
- [Real](#) [theta](#) () const
- [Real](#) [vega](#) () const
- [Real](#) [rho](#) () const
- [Real](#) [dividendRho](#) () const

Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

Protected Attributes

- [Real](#) [delta_](#)
- [Real](#) [gamma_](#)
- [Real](#) [theta_](#)
- [Real](#) [vega_](#)
- [Real](#) [rho_](#)
- [Real](#) [dividendRho_](#)
- std::vector< boost::shared_ptr< [BlackScholesProcess](#) > > [blackScholesProcesses_](#)
- [Matrix](#) [correlation_](#)

7.351.2 Member Function Documentation

7.351.2.1 void [setupArguments](#) ([Arguments](#) *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

Reimplemented in [BasketOption](#).

7.351.2.2 void [setupExpired](#) () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [Instrument](#).

7.351.2.3 void [performCalculations](#) () const [protected, virtual]

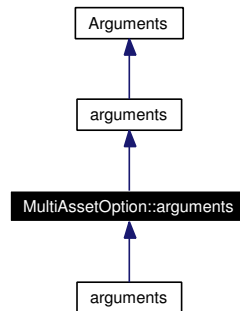
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

7.352 MultiAssetOption::arguments Class Reference

```
#include <ql/Instruments/multiassetoption.hpp>
```

Inheritance diagram for MultiAssetOption::arguments:



7.352.1 Detailed Description

Arguments for multi-asset option calculation

Public Member Functions

- `void validate () const`

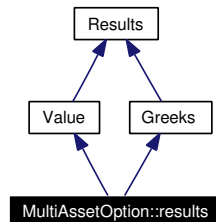
Public Attributes

- `std::vector< boost::shared_ptr< BlackScholesProcess > > blackScholesProcesses`
- `Matrix correlation`

7.353 MultiAssetOption::results Class Reference

```
#include <ql/Instruments/multiassetoption.hpp>
```

Inheritance diagram for MultiAssetOption::results:



7.353.1 Detailed Description

Results from multi-asset option calculation

Public Member Functions

- void `reset ()`

7.354 MultiCubicSpline Class Template Reference

```
#include <ql/Math/multicubicspline.hpp>
```

7.354.1 Detailed Description

```
template<Size i> class QuantLib::MultiCubicSpline< i >
```

Tests

interpolated values are checked against the original function.

Todo

- fix it for Borland compilation
 - allow extrapolation as for the other interpolations
 - investigate if and how to implement Hyman filters and different boundary conditions

Bug

- a) cannot interpolate at the grid points on the boundary surface of the N-dimensional region
- b) it does not compile under Borland

Public Types

- typedef c_splint::argument_type **argument_type**
- typedef c_splint::result_type **result_type**
- typedef c_splint::data_table **data_table**
- typedef c_splint::return_type **return_type**
- typedef c_splint::output_data **output_data**
- typedef c_splint::dimensions **dimensions**
- typedef c_splint::data **data**

Public Member Functions

- **MultiCubicSpline** (const SplineGrid &grid, const data_table &y, const std::vector< bool > &ae=std::vector< bool >(20, false))
- result_type **operator()** (const argument_type &x) const
- void **set_shared_increments** () const
- void **set_shared_coefficients** (const argument_type &x) const

7.355 MultiPath Class Reference

```
#include <ql/MonteCarlo/multipath.hpp>
```

7.355.1 Detailed Description

Correlated multiple asset paths.

[MultiPath](#) 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\[j,i\]](#)

Todo

rename 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)

7.356 MultiPathGenerator Class Template Reference

```
#include <ql/MonteCarlo/multipathgenerator.hpp>
```

7.356.1 Detailed Description

```
template<class GSG> class QuantLib::MultiPathGenerator< GSG >
```

Generates a multipath from a random number generator.

RSG is a sample generator which returns a random sequence. It must have the minimal interface:

```
RSG {  
    Sample<Array> next();  
};
```

Todo

why store correlation matrix rather than covariance?

Public Types

- typedef [Sample](#)< [MultiPath](#) > **sample_type**

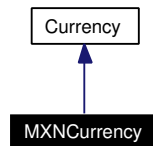
Public Member Functions

- **MultiPathGenerator** (const std::vector< boost::shared_ptr< [StochasticProcess](#) > > &diffusionProcs, const [Matrix](#) &correlation, const [TimeGrid](#) &timeGrid, GSG generator, bool brownianBridge=false)
- const [sample_type](#) & **next** () const
- const [sample_type](#) & **antithetic** () const

7.357 MXNCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for MXNCurrency:



7.357.1 Detailed Description

Mexican peso.

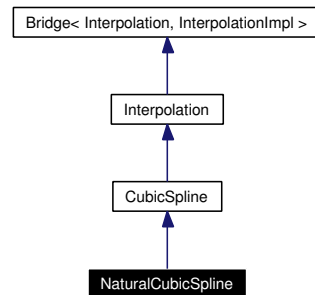
The ISO three-letter code is MXN; the numeric code is 484. It is divided in 100 centavos.

ingroup currencies

7.358 NaturalCubicSpline Class Reference

```
#include <ql/Math/cubicspline.hpp>
```

Inheritance diagram for NaturalCubicSpline:



7.358.1 Detailed Description

Cubic spline with null second derivative at end points

Public Member Functions

- `template<class I1, class I2> NaturalCubicSpline (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)`

7.358.2 Constructor & Destructor Documentation

7.358.2.1 [NaturalCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

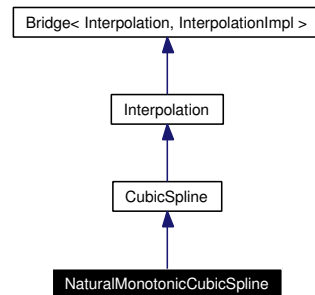
Precondition:

the *x* values must be sorted.

7.359 NaturalMonotonicCubicSpline Class Reference

#include <ql/Math/cubicspline.hpp>

Inheritance diagram for NaturalMonotonicCubicSpline:



7.359.1 Detailed Description

Natural cubic spline with monotonicity constraint.

Public Member Functions

- template<class I1, class I2> [NaturalMonotonicCubicSpline](#) (const I1 &xBegin, const I1 &xEnd, const I2 &yBegin)

7.359.2 Constructor & Destructor Documentation

- 7.359.2.1 [NaturalMonotonicCubicSpline](#) (const I1 & *xBegin*, const I1 & *xEnd*, const I2 & *yBegin*)

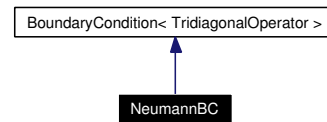
Precondition:

the *x* values must be sorted.

7.360 NeumannBC Class Reference

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Inheritance diagram for NeumannBC:



7.360.1 Detailed Description

Neumann boundary condition (i.e., constant derivative).

Warning:

The value passed must not be the value of the derivative. Instead, it must be comprehensive of the grid step between the first two points—i.e., it must be the difference between $f[0]$ and $f[1]$.

Todo

generalize to time-dependent conditions.

Public Member Functions

- **NeumannBC** ([Real](#) value, [Side](#) side)
- void **applyBeforeApplying** ([TridiagonalOperator](#) &) const
- void **applyAfterApplying** ([Array](#) &) const
- void **applyBeforeSolving** ([TridiagonalOperator](#) &, [Array](#) &rhs) const
- void **applyAfterSolving** ([Array](#) &) const
- void **setTime** ([Time](#))

7.360.2 Member Function Documentation

7.360.2.1 void setTime ([Time](#)) [virtual]

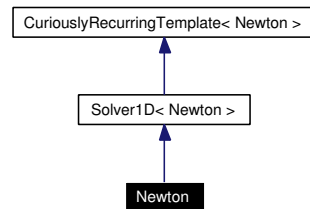
This method sets the current time for time-dependent boundary conditions.

Implements [BoundaryCondition< TridiagonalOperator >](#).

7.361 Newton Class Reference

```
#include <ql/Solvers1D/newton.hpp>
```

Inheritance diagram for Newton:



7.361.1 Detailed Description

Newton 1-D solver

Note:

This solver requires that the passed function object implement a method `Real derivative(Real)`.

Tests

the correctness of the returned values is tested by checking them against known good results.

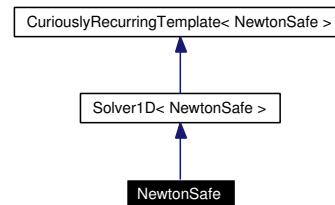
Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

7.362 NewtonSafe Class Reference

```
#include <ql/Solvers1D/newtonsafe.hpp>
```

Inheritance diagram for NewtonSafe:



7.362.1 Detailed Description

safe Newton 1-D solver

Note:

This solver requires that the passed function object implement a method `Real derivative(Real)`.

Tests

the correctness of the returned values is tested by checking them against known good results.

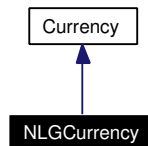
Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

7.363 NLGCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for NLGCurrency:



7.363.1 Detailed Description

Dutch guilder.

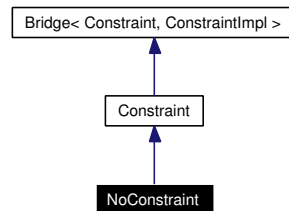
The ISO three-letter code is NLG; the numeric code is 528. It is divided in 100 cents.

ingroup currencies

7.364 NoConstraint Class Reference

```
#include <ql/Optimization/constraint.hpp>
```

Inheritance diagram for NoConstraint:



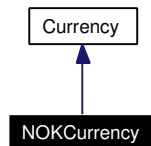
7.364.1 Detailed Description

No constraint.

7.365 NOKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for NOKCurrency:



7.365.1 Detailed Description

Norwegian krone.

The ISO three-letter code is NOK; the numeric code is 578. It is divided in 100 øre.

ingroup currencies

7.366 NonLinearLeastSquare Class Reference

```
#include <ql/Optimization/leastsquare.hpp>
```

7.366.1 Detailed Description

Non-linear least-square method.

Using a given optimization algorithm (default is conjugate gradient),

$$\min\{r(x) : x \in R^n\}$$

where $r(x) = |f(x)|^2$ is the Euclidean norm of $f(x)$ for some vector-valued function f from R^n to R^m ,

$$f = (f_1, \dots, f_m)$$

with $f_i(x) = b_i - \phi(x, t_i)$ where b is the vector of target data and ϕ is a scalar function.

Assuming the differentiability of f , the gradient of r is defined by

$$\text{grad}r(x) = f'(x)^t \cdot f(x)$$

Public Member Functions

- [NonLinearLeastSquare](#) ([Constraint](#) &c, [Real](#) accuracy=1e-4, [Size](#) maxiter=100)
Default constructor.
- [NonLinearLeastSquare](#) ([Constraint](#) &c, [Real](#) accuracy, [Size](#) maxiter, [boost::shared_ptr](#)<[OptimizationMethod](#)> om)
Default constructor.
- [~NonLinearLeastSquare](#) ()
Destructor.
- [Array](#) & [perform](#) ([LeastSquareProblem](#) &lsProblem)
Solve least square problem using numerix solver.
- void [setInitialValue](#) (const [Array](#) &initialValue)
- [Array](#) & [results](#) ()
return the results
- [Real](#) [residualNorm](#) ()
return the least square residual norm
- [Real](#) [lastValue](#) ()
return last function value
- [Integer](#) [exitFlag](#) ()
return exit flag
- [Integer](#) [iterationsNumber](#) ()
return the performed number of iterations

7.367 NormalDistribution Class Reference

```
#include <ql/Math/normaldistribution.hpp>
```

7.367.1 Detailed Description

Normal distribution function.

Given x , it returns its probability in a Gaussian normal distribution. It provides the first derivative too.

Tests

the correctness of the returned value is tested by checking it against numerical calculations. Cross-checks are also performed against the [CumulativeNormalDistribution](#) and [InverseCumulativeNormal](#) classes.

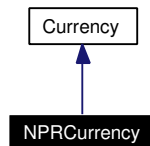
Public Member Functions

- **NormalDistribution** ([Real](#) average=0.0, [Real](#) sigma=1.0)
- **Real operator()** ([Real](#) x) const
- **Real derivative** ([Real](#) x) const

7.368 NPRCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for NPRCurrency:



7.368.1 Detailed Description

Nepal rupee.

The ISO three-letter code is NPR; the numeric code is 524. It is divided in 100 paise.

ingroup currencies

7.369 Null Class Template Reference

```
#include <ql/null.hpp>
```

7.369.1 Detailed Description

```
template<class Type> class QuantLib::Null< Type >
```

template class providing a null value for a given type.

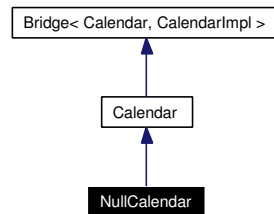
Public Member Functions

- `operator Type () const`

7.370 NullCalendar Class Reference

```
#include <ql/Calendars/nullcalendar.hpp>
```

Inheritance diagram for NullCalendar:



7.370.1 Detailed Description

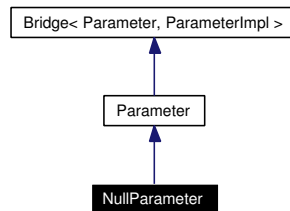
Calendar for reproducing theoretical calculations.

This calendar has no holidays. It ensures that dates at whole-month distances have the same day of month.

7.371 NullParameter Class Reference

```
#include <ql/ShortRateModels/parameter.hpp>
```

Inheritance diagram for NullParameter:



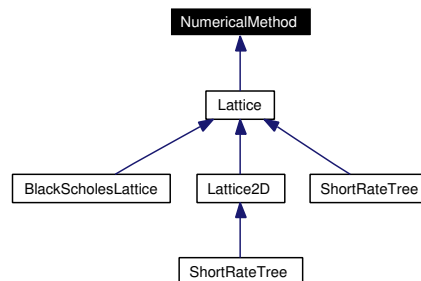
7.371.1 Detailed Description

Parameter which is always zero $a(t) = 0$

7.372 NumericalMethod Class Reference

```
#include <ql/numericalmethod.hpp>
```

Inheritance diagram for NumericalMethod:



7.372.1 Detailed Description

Numerical method (tree, finite-differences) base class.

Public Member Functions

- **NumericalMethod** (const [TimeGrid](#) &timeGrid)

Inspectors

- const [TimeGrid](#) & **timeGrid** () const

Numerical method interface

*These methods are to be used by discretized assets and must be overridden by developers implementing numerical methods. Users are advised to use the corresponding methods of *DiscretizedAsset* instead.*

- virtual void **initialize** ([DiscretizedAsset](#) &, [Time](#) time) const =0
initialize an asset at the given time.
- virtual void **rollback** ([DiscretizedAsset](#) &, [Time](#) to) const =0
- virtual void **partialRollback** ([DiscretizedAsset](#) &, [Time](#) to) const =0
- virtual [Real](#) **presentValue** ([DiscretizedAsset](#) &)=0
computes the present value of an asset.

Protected Attributes

- [TimeGrid](#) t_

7.372.2 Member Function Documentation

7.372.2.1 virtual void rollback ([DiscretizedAsset](#) &, [Time](#) to) const [pure virtual]

Roll back an asset until the given time, performing any needed adjustment.

Implemented in [Lattice](#).

7.372.2.2 `virtual void partialRollback (DiscretizedAsset &, Time to) const` [pure virtual]

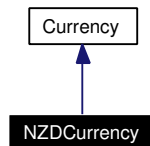
Roll back an asset until the given time, but do not perform the final adjustment.

Implemented in [Lattice](#).

7.373 NZDCurrency Class Reference

```
#include <ql/Currencies/oceania.hpp>
```

Inheritance diagram for NZDCurrency:



7.373.1 Detailed Description

New Zealand dollar.

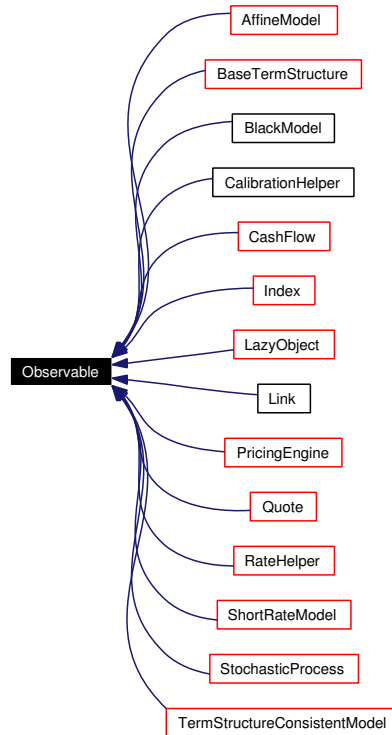
The ISO three-letter code is NZD; the numeric code is 554. It is divided in 100 cents.

ingroup currencies

7.374 Observable Class Reference

```
#include <ql/Patterns/observable.hpp>
```

Inheritance diagram for Observable:



7.374.1 Detailed Description

Object that notifies its changes to a set of observables.

Public Member Functions

- void [notifyObservers](#) ()

Friends

- class `Observer`

7.374.2 Member Function Documentation

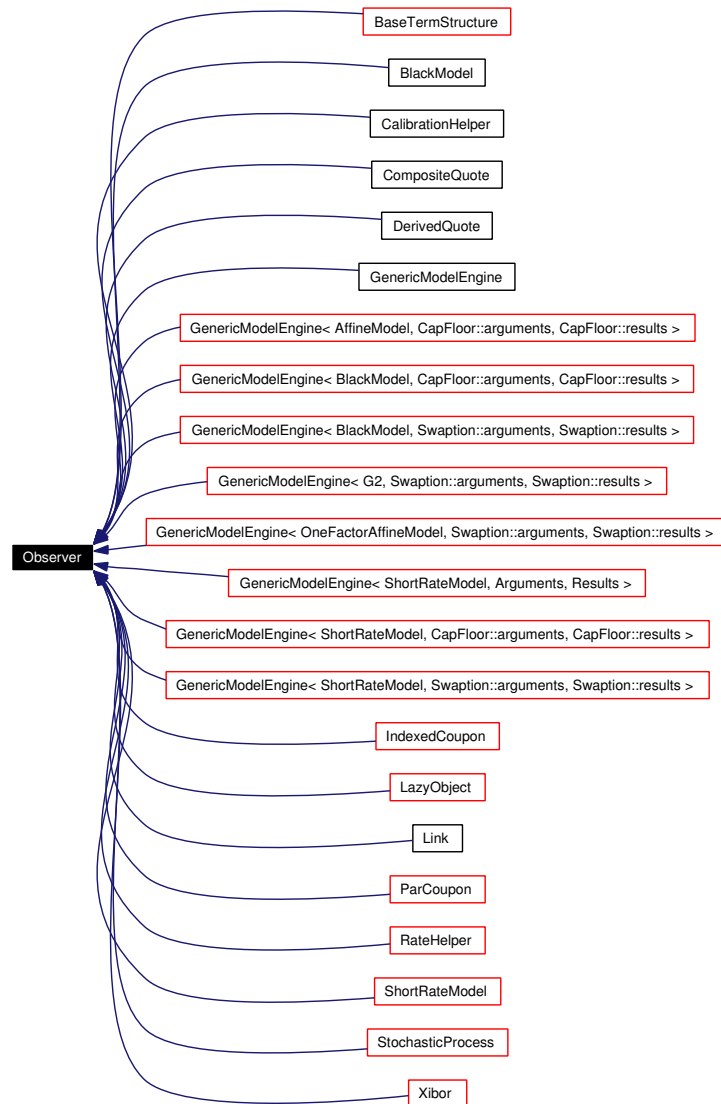
7.374.2.1 void notifyObservers ()

This method should be called at the end of non-const methods or when the programmer desires to notify any changes.

7.375 Observer Class Reference

```
#include <ql/Patterns/observable.hpp>
```

Inheritance diagram for Observer:



7.375.1 Detailed Description

Object that gets notified when a given observable changes.

Public Member Functions

- **Observer** (const **Observer** &)
- **Observer** & **operator=** (const **Observer** &)
- template<class T> void **registerWith** (const boost::shared_ptr< T > &h)

- `template<class T> void unregisterWith (const boost::shared_ptr< T > &h)`
- `virtual void update ()=0`

7.375.2 Member Function Documentation

7.375.2.1 `virtual void update ()` [pure virtual]

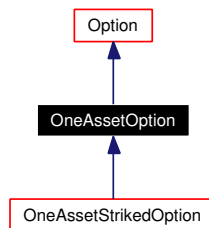
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implemented in [BaseTermStructure](#), [IndexedCoupon](#), [ParCoupon](#), [Xibor](#), [LazyObject](#), [BlackModel](#), [GenericModelEngine](#), [LatticeShortRateModelEngine](#), [DerivedQuote](#), [CompositeQuote](#), [Link](#), [CalibrationHelper](#), [ShortRateModel](#), [StochasticProcess](#), [BlackScholesProcess](#), [AffineTermStructure](#), [ExtendedDiscountCurve](#), [PiecewiseFlatForward](#), [RateHelper](#), [CapVolatilityVector](#), [GenericModelEngine< ShortRateModel, Arguments, Results >](#), [GenericModelEngine< BlackModel, CapFloor::arguments, CapFloor::results >](#), [GenericModelEngine< OneFactorAffineModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< G2, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< AffineModel, CapFloor::arguments, CapFloor::results >](#), [GenericModelEngine< BlackModel, Swaption::arguments, Swaption::results >](#), [GenericModelEngine< ShortRateModel, CapFloor::arguments, CapFloor::results >](#), [GenericModelEngine< ShortRateModel, Swaption::arguments, Swaption::results >](#), [LatticeShortRateModelEngine< CapFloor::arguments, CapFloor::results >](#), and [LatticeShortRateModelEngine< Swaption::arguments, Swaption::results >](#).

7.376 OneAssetOption Class Reference

```
#include <ql/Instruments/oneassetoption.hpp>
```

Inheritance diagram for OneAssetOption:



7.376.1 Detailed Description

Base class for options on a single asset.

Public Member Functions

- **OneAssetOption** (const boost::shared_ptr< [BlackScholesProcess](#) > &, const boost::shared_ptr< [Payoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- [Volatility](#) [impliedVolatility](#) ([Real](#) price, [Real](#) accuracy=1.0e-4, [Size](#) maxEvaluations=100, [Volatility](#) minVol=QL_MIN_VOLATILITY, [Volatility](#) maxVol=QL_MAX_VOLATILITY) const
- void [setupArguments](#) ([Arguments](#) *) const

Instrument interface

- bool [isExpired](#) () const
returns whether the instrument is still tradable.

greeks

- [Real](#) [delta](#) () const
- [Real](#) [deltaForward](#) () const
- [Real](#) [elasticity](#) () const
- [Real](#) [gamma](#) () const
- [Real](#) [theta](#) () const
- [Real](#) [thetaPerDay](#) () const
- [Real](#) [vega](#) () const
- [Real](#) [rho](#) () const
- [Real](#) [dividendRho](#) () const
- [Real](#) [itmCashProbability](#) () const

Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

Protected Attributes

- [Real](#) `delta_`
- [Real](#) `deltaForward_`
- [Real](#) `elasticity_`
- [Real](#) `gamma_`
- [Real](#) `theta_`
- [Real](#) `thetaPerDay_`
- [Real](#) `vega_`
- [Real](#) `rho_`
- [Real](#) `dividendRho_`
- [Real](#) `itmCashProbability_`
- `boost::shared_ptr< BlackScholesProcess > blackScholesProcess_`

7.376.2 Member Function Documentation

7.376.2.1 [Volatility](#) `impliedVolatility (Real price, Real accuracy = 1.0e-4, Size maxEvaluations = 100, Volatility minVol = QL_MIN_VOLATILITY, Volatility maxVol = QL_MAX_VOLATILITY) const`

Warning:

currently, this method returns the Black-Scholes implied volatility. It will give inconsistent results if the pricing was performed with any other methods (such as jump-diffusion models.) options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases the calculation can fail and the result (if any) is almost meaningless. Another possible source of failure is to have a target value that is not attainable with any volatility, e.g., a target value lower than the intrinsic value in the case of American options.

7.376.2.2 `void setupArguments (Arguments *) const` [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

Reimplemented in [ContinuousAveragingAsianOption](#), [DiscreteAveragingAsianOption](#), [BarrierOption](#), [CliquetOption](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [OneAssetStrikedOption](#), [QuantoForwardVanillaOption](#), and [QuantoVanillaOption](#).

7.376.2.3 `void setupExpired () const` [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [Instrument](#).

Reimplemented in [QuantoVanillaOption](#).

7.376.2.4 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

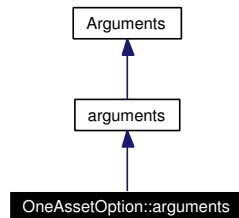
Reimplemented from [Instrument](#).

Reimplemented in [BarrierOption](#), [ForwardVanillaOption](#), [OneAssetStrikedOption](#), and [QuantoVanillaOption](#).

7.377 OneAssetOption::arguments Class Reference

```
#include <ql/Instruments/oneassetoption.hpp>
```

Inheritance diagram for OneAssetOption::arguments:



7.377.1 Detailed Description

Arguments for single-asset option calculation

Public Member Functions

- void **validate** () const

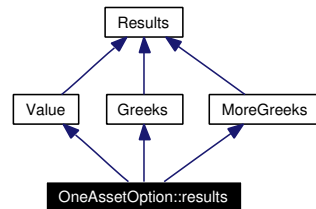
Public Attributes

- boost::shared_ptr< [BlackScholesProcess](#) > **blackScholesProcess**

7.378 OneAssetOption::results Class Reference

```
#include <ql/Instruments/oneassetoption.hpp>
```

Inheritance diagram for OneAssetOption::results:



7.378.1 Detailed Description

Results from single-asset option calculation

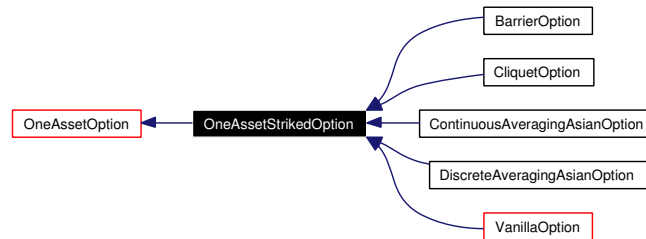
Public Member Functions

- `void reset ()`

7.379 OneAssetStrikedOption Class Reference

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Inheritance diagram for OneAssetStrikedOption:



7.379.1 Detailed Description

Base class for options on a single asset with striked payoff.

Public Member Functions

- **OneAssetStrikedOption** (const boost::shared_ptr< [BlackScholesProcess](#) > &stochProc, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())
- void [setupArguments](#) ([Arguments](#) *) const

greeks

- [Real](#) [strikeSensitivity](#) () const

Protected Member Functions

- void [performCalculations](#) () const

Protected Attributes

- [Real](#) [strikeSensitivity_](#)

7.379.2 Member Function Documentation

7.379.2.1 void [setupArguments](#) ([Arguments](#) *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetOption](#).

Reimplemented in [ContinuousAveragingAsianOption](#), [DiscreteAveragingAsianOption](#), [BarrierOption](#), [CliquetOption](#), [DividendVanillaOption](#), [ForwardVanillaOption](#), [QuantoForwardVanillaOption](#), and [QuantoVanillaOption](#).

7.379.2.2 void performCalculations () const [protected, virtual]

In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

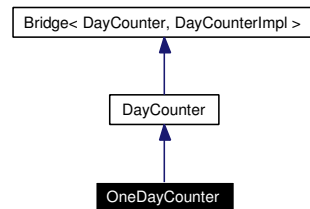
Reimplemented from [OneAssetOption](#).

Reimplemented in [BarrierOption](#), [ForwardVanillaOption](#), and [QuantoVanillaOption](#).

7.380 OneDayCounter Class Reference

```
#include <ql/DayCounters/one.hpp>
```

Inheritance diagram for OneDayCounter:



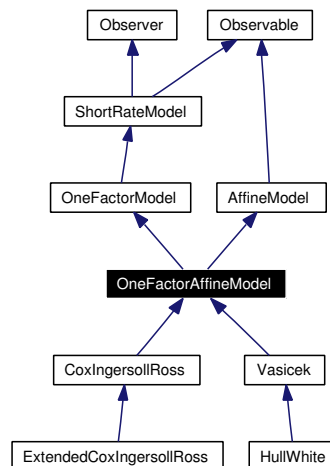
7.380.1 Detailed Description

1/1 day count convention

7.381 OneFactorAffineModel Class Reference

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Inheritance diagram for OneFactorAffineModel:



7.381.1 Detailed Description

Single-factor affine base class.

Single-factor models with an analytical formula for discount bonds should inherit from this class. They must then implement the functions $A(t, T)$ and $B(t, T)$ such that

$$P(t, T, r_t) = A(t, T)e^{-B(t, T)r_t}.$$

Public Member Functions

- **OneFactorAffineModel** ([Size](#) nArguments)
- **Real discountBond** ([Time](#) now, [Time](#) maturity, [Rate](#) rate) const
- **DiscountFactor discount** ([Time](#) t) const

Implied discount curve.

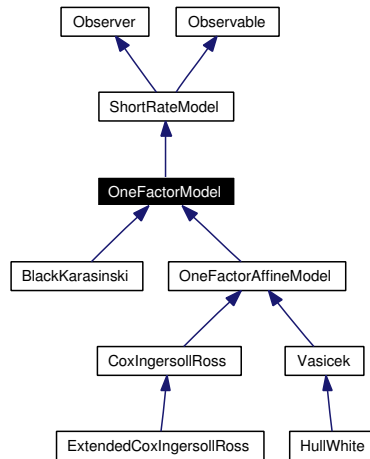
Protected Member Functions

- virtual **Real A** ([Time](#) t, [Time](#) T) const =0
- virtual **Real B** ([Time](#) t, [Time](#) T) const =0

7.382 OneFactorModel Class Reference

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Inheritance diagram for OneFactorModel:



7.382.1 Detailed Description

Single-factor short-rate model abstract class.

Public Member Functions

- **OneFactorModel** ([Size](#) nArguments)
- virtual boost::shared_ptr< [ShortRateDynamics](#) > [dynamics](#) () const =0
returns the short-rate dynamics
- virtual boost::shared_ptr< [Lattice](#) > [tree](#) (const [TimeGrid](#) &grid) const
Return by default a trinomial recombining tree.

7.383 OneFactorModel::ShortRateDynamics Class Reference

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

7.383.1 Detailed Description

Base class describing the short-rate dynamics.

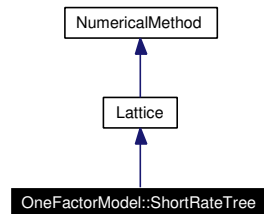
Public Member Functions

- **ShortRateDynamics** (const boost::shared_ptr< [StochasticProcess](#) > &process)
- virtual [Real](#) [variable](#) ([Time](#) t, [Rate](#) r) const =0
Compute state variable from short rate.
- virtual [Rate](#) [shortRate](#) ([Time](#) t, [Real](#) variable) const =0
Compute short rate from state variable.
- const boost::shared_ptr< [StochasticProcess](#) > & [process](#) ()
Returns the risk-neutral dynamics of the state variable.

7.384 OneFactorModel::ShortRateTree Class Reference

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Inheritance diagram for OneFactorModel::ShortRateTree:



7.384.1 Detailed Description

Recombining trinomial tree discretizing the state variable.

Public Member Functions

- [ShortRateTree](#) (const boost::shared_ptr< [Tree](#) > &tree, const boost::shared_ptr< [ShortRateDynamics](#) > &dynamics, const [TimeGrid](#) &timeGrid)
Plain tree build-up from short-rate dynamics.
- [ShortRateTree](#) (const boost::shared_ptr< [Tree](#) > &tree, const boost::shared_ptr< [ShortRateDynamics](#) > &dynamics, const boost::shared_ptr< [TermStructureFittingParameter::NumericalImpl](#) > &phi, const [TimeGrid](#) &timeGrid)
Tree build-up + numerical fitting to term-structure.
- [Size](#) [size](#) ([Size](#) i) const
- [DiscountFactor](#) [discount](#) ([Size](#) i, [Size](#) index) const
Discount factor at time t_i and node indexed by index.

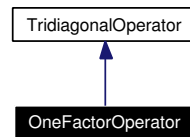
Protected Member Functions

- [Size](#) [descendant](#) ([Size](#) i, [Size](#) index, [Size](#) branch) const
Tree properties.
- [Real](#) [probability](#) ([Size](#) i, [Size](#) index, [Size](#) branch) const

7.385 OneFactorOperator Class Reference

```
#include <ql/FiniteDifferences/onefactoroperator.hpp>
```

Inheritance diagram for OneFactorOperator:



7.385.1 Detailed Description

Interest-rate single factor model differential operator.

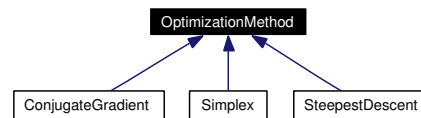
Public Member Functions

- **OneFactorOperator** (const [Array](#) &grid, const boost::shared_ptr< [OneFactorModel::ShortRateDynamics](#) > &)

7.386 OptimizationMethod Class Reference

```
#include <ql/Optimization/method.hpp>
```

Inheritance diagram for OptimizationMethod:



7.386.1 Detailed Description

Abstract class for constrained optimization method.

Public Member Functions

- void **setInitialValue** (const **Array** &initialValue)
Set initial value.
- void **setEndCriteria** (const **EndCriteria** &endCriteria)
Set optimization end criteria.
- **Integer** & **iterationNumber** () const
current iteration number
- **EndCriteria** & **endCriteria** () const
optimization end criteria
- **Integer** & **functionEvaluation** () const
number of evaluation of cost function
- **Integer** & **gradientEvaluation** () const
number of evaluation of cost function gradient
- **Real** & **functionValue** () const
value of cost function
- **Real** & **gradientNormValue** () const
value of cost function gradient norm
- **Array** & **x** () const
current value of the local minimum
- **Array** & **searchDirection** () const
current value of the search direction
- virtual void **minimize** (const **Problem** &P) const =0
minimize the optimization problem P

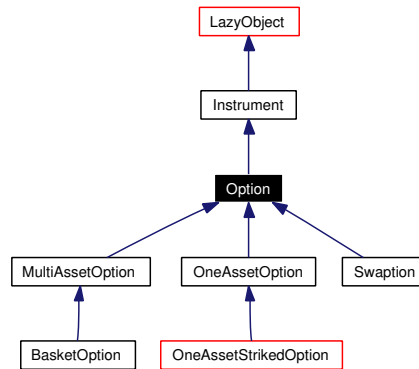
Protected Attributes

- [Array initialValue_](#)
initial value of unknowns
- [Integer iterationNumber_](#)
current iteration step in the Optimization process
- [EndCriteria endCriteria_](#)
optimization end criteria
- [Integer functionEvaluation_](#)
number of evaluation of cost function and its gradient
- [Integer gradientEvaluation_](#)
number of evaluation of cost function and its gradient
- [Real functionValue_](#)
function and gradient norm values of the last step
- [Real 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

7.387 Option Class Reference

```
#include <ql/option.hpp>
```

Inheritance diagram for Option:



7.387.1 Detailed Description

base option class

Public Types

- enum Type { Call, Put }

Public Member Functions

- Option** (const boost::shared_ptr< [Payoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())

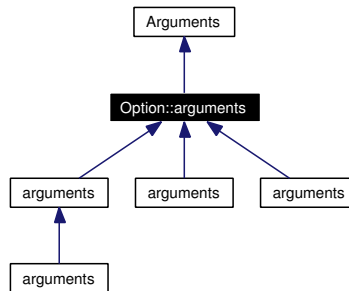
Protected Attributes

- boost::shared_ptr< [Payoff](#) > **payoff_**
- boost::shared_ptr< [Exercise](#) > **exercise_**

7.388 Option::arguments Class Reference

```
#include <ql/option.hpp>
```

Inheritance diagram for Option::arguments:



7.388.1 Detailed Description

basic option arguments

Todo

- remove `std::vector<Time> stoppingTimes`
- how to handle strike-less option (asian average strike, forward, etc.)?

Public Member Functions

- void **validate** () const

Public Attributes

- boost::shared_ptr< [Payoff](#) > **payoff**
- boost::shared_ptr< [Exercise](#) > **exercise**
- std::vector< [Time](#) > **stoppingTimes**

7.389 OptionTypeFormatter Class Reference

```
#include <ql/option.hpp>
```

7.389.1 Detailed Description

format option type for output

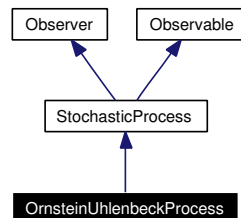
Static Public Member Functions

- `std::string toString (Option::Type type)`

7.390 OrnsteinUhlenbeckProcess Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for OrnsteinUhlenbeckProcess:



7.390.1 Detailed Description

Ornstein-Uhlenbeck process class.

This class describes the Ornstein-Uhlenbeck process governed by

$$dx = -ax_t dt + \sigma dW_t.$$

Public Member Functions

- **OrnsteinUhlenbeckProcess** ([Real](#) speed, [Volatility](#) vol, [Real](#) x0=0.0)

StochasticProcess interface

- [Real](#) **x0** () const
returns the initial value of the state variable
- [Real](#) **drift** ([Time](#) t, [Real](#) x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- [Real](#) **diffusion** ([Time](#) t, [Real](#) x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- [Real](#) **expectation** ([Time](#) t0, [Real](#) x0, [Time](#) dt) const
- [Real](#) **variance** ([Time](#) t0, [Real](#) x0, [Time](#) dt) const

7.390.2 Member Function Documentation

7.390.2.1 [Real](#) expectation ([Time](#) t0, [Real](#) x0, [Time](#) dt) const [virtual]

returns the expectation $E(x_{t_0+\Delta t} | x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

7.390.2.2 Real variance (Time t_0 , Real x_0 , Time dt) const [virtual]

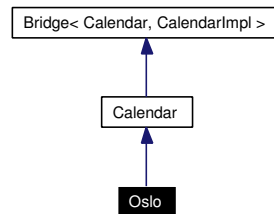
returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented from [StochasticProcess](#).

7.391 Oslo Class Reference

```
#include <ql/Calendars/oslo.hpp>
```

Inheritance diagram for Oslo:



7.391.1 Detailed Description

Oslo calendar

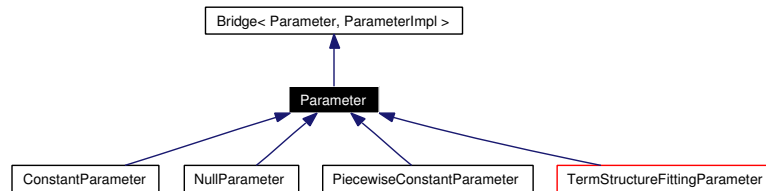
Holidays:

- Saturdays
- Sundays
- Holy Thursday
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- New Year's Day, January 1st
- May Day, May 1st
- National Independence Day, May 17st
- Christmas, December 25th
- Boxing Day, December 26th

7.392 Parameter Class Reference

#include <ql/ShortRateModels/parameter.hpp>

Inheritance diagram for Parameter:



7.392.1 Detailed Description

Base class for model arguments.

Public Member Functions

- const [Array](#) & **params** () const
- void **setParam** ([Size](#) i, [Real](#) x)
- bool **testParams** (const [Array](#) ¶ms) const
- [Size](#) **size** () const
- [Real](#) **operator()** ([Time](#) t) const
- const boost::shared_ptr< [ParameterImpl](#) > & **implementation** () const

Protected Member Functions

- **Parameter** ([Size](#) size, const boost::shared_ptr< [ParameterImpl](#) > &impl, const [Constraint](#) &constraint)

Protected Attributes

- [Array](#) **params_**
- [Constraint](#) **constraint_**

7.393 ParameterImpl Class Reference

```
#include <ql/ShortRateModels/parameter.hpp>
```

7.393.1 Detailed Description

Base class for model parameter implementation.

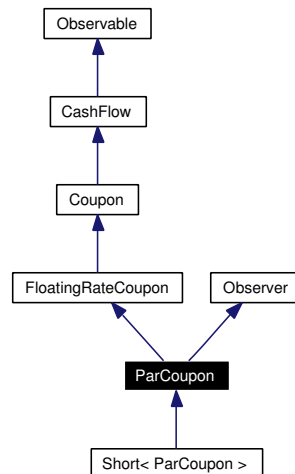
Public Member Functions

- virtual [Real](#) value (const [Array](#) ¶ms, [Time](#) t) const =0

7.394 ParCoupon Class Reference

```
#include <ql/CashFlows/parcoupon.hpp>
```

Inheritance diagram for ParCoupon:



7.394.1 Detailed Description

coupon at par on a term structure

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **ParCoupon** ([Real](#) nominal, const [Date](#) &paymentDate, const boost::shared_ptr< [Xibor](#) > &index, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())

CashFlow interface

- [Real](#) amount () const
returns the amount of the cash flow

Coupon interface

- [DayCounter](#) dayCounter () const
day counter for accrual calculation

FloatingRateCoupon interface

- [Rate indexFixing](#) () const
fixing of the underlying index
- [Date fixingDate](#) () const
fixing date
- [Rate fixing](#) () const

Inspectors

- const boost::shared_ptr< [Xibor](#) > & [index](#) () const

Observer interface

- void [update](#) ()

Visitability

- virtual void [accept](#) ([AcyclicVisitor](#) &)

7.394.2 Member Function Documentation

7.394.2.1 [Real](#) amount () const [virtual]

returns the amount of the cash flow

Note:

The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

Reimplemented in [Short< ParCoupon >](#).

7.394.2.2 [Rate](#) fixing () const [virtual]

Deprecated

use [rate\(\)](#) instead

Implements [FloatingRateCoupon](#).

7.394.2.3 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.395 Path Class Reference

```
#include <ql/MonteCarlo/path.hpp>
```

7.395.1 Detailed Description

single factor random walk path pricer

Todo

should [Path](#) 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

- [Real](#) **operator[]** ([Size](#) 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** ()

7.396 PathGenerator Class Template Reference

```
#include <ql/MonteCarlo/pathgenerator.hpp>
```

7.396.1 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 boost::shared_ptr< [StochasticProcess](#) > &diffProcess, [Time](#) length, [Size](#) timeSteps, const GSG &generator, bool brownianBridge)
- **PathGenerator** (const boost::shared_ptr< [StochasticProcess](#) > &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

7.397 PathPricer Class Template Reference

```
#include <ql/MonteCarlo/pathpricer.hpp>
```

7.397.1 Detailed Description

```
template<class PathType, class ValueType = Real> class QuantLib::PathPricer< PathType,
ValueType >
```

base class for path pricers

Returns the value of an option on a given path.

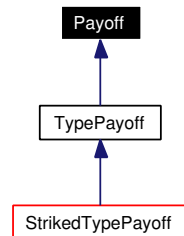
Public Member Functions

- virtual ValueType **operator()** (const PathType &path) const =0

7.398 Payoff Class Reference

```
#include <ql/payoff.hpp>
```

Inheritance diagram for Payoff:



7.398.1 Detailed Description

Base class for option payoffs.

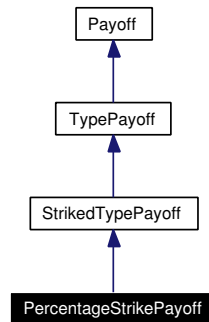
Public Member Functions

- virtual [Real](#) operator() ([Real](#) price) const =0

7.399 PercentageStrikePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for PercentageStrikePayoff:



7.399.1 Detailed Description

Payoff with strike expressed as percentage

Public Member Functions

- **PercentageStrikePayoff** (Option::Type type, [Real](#) moneyness)
- **Real operator()** ([Real](#) price) const

7.400 Period Class Reference

```
#include <ql/date.hpp>
```

7.400.1 Detailed Description

Time period described by a number of a given time unit.

Public Member Functions

- **Period** ([Integer](#) n, [TimeUnit](#) units)
- [Integer](#) **length** () const
- [TimeUnit](#) **units** () const

Related Functions

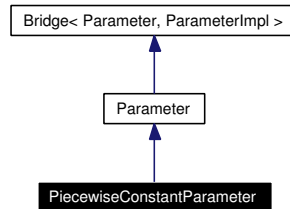
(Note that these are not member functions.)

- [Period](#) **operator** * ([Integer](#) n, [TimeUnit](#) units)
- [Period](#) **operator** * ([TimeUnit](#) units, [Integer](#) n)
- bool **operator**< (const [Period](#) &, const [Period](#) &)
- bool **operator**== (const [Period](#) &, const [Period](#) &)
- bool **operator**!= (const [Period](#) &, const [Period](#) &)
- bool **operator**> (const [Period](#) &, const [Period](#) &)
- bool **operator**<= (const [Period](#) &, const [Period](#) &)
- bool **operator**>= (const [Period](#) &, const [Period](#) &)

7.401 PiecewiseConstantParameter Class Reference

```
#include <ql/ShortRateModels/parameter.hpp>
```

Inheritance diagram for PiecewiseConstantParameter:



7.401.1 Detailed Description

Piecewise-constant parameter.

$a(t) = a_i$ if $t_{i-1} \leq t < t_i$. This kind of parameter is usually used to enhance the fitting of a model

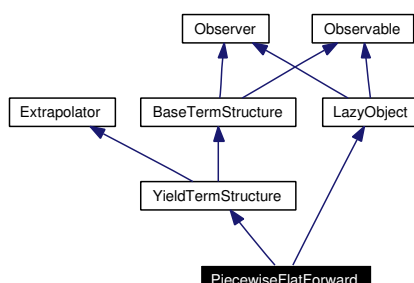
Public Member Functions

- **PiecewiseConstantParameter** (const std::vector< [Time](#) > ×)

7.402 PiecewiseFlatForward Class Reference

```
#include <ql/TermStructures/piecewiseflatforward.hpp>
```

Inheritance diagram for PiecewiseFlatForward:



7.402.1 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](#) 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.

Tests

- the correctness of the returned values is tested by checking them against the original inputs.
- the observability of the term structure is tested.

Public Member Functions

Constructors

- PiecewiseFlatForward** (const [Date](#) &todayDate, const [Date](#) &referenceDate, const std::vector< boost::shared_ptr< [RateHelper](#) > > &instruments, const [DayCounter](#) &dayCounter, [Real](#) accuracy=1.0e-12)
- PiecewiseFlatForward** (const [Date](#) &todayDate, const std::vector< [Date](#) > &dates, const std::vector< [Rate](#) > &forwards, const [DayCounter](#) &dayCounter)
- PiecewiseFlatForward** (const [Date](#) &referenceDate, const std::vector< boost::shared_ptr< [RateHelper](#) > > &instruments, const [DayCounter](#) &dayCounter, [Real](#) accuracy=1.0e-12)
- PiecewiseFlatForward** ([Integer](#) settlementDays, const [Calendar](#) &calendar, const std::vector< boost::shared_ptr< [RateHelper](#) > > &instruments, const [DayCounter](#) &dayCounter, [Real](#) accuracy=1.0e-12)

- [PiecewiseFlatForward](#) (const std::vector< [Date](#) > &dates, const std::vector< [Rate](#) > &forwards, const [DayCounter](#) &dayCounter)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- 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

Observer interface

- void [update](#) ()

Protected Member Functions

- [Rate](#) [zeroYieldImpl](#) ([Time](#)) const
zero-yield calculation
- [DiscountFactor](#) [discountImpl](#) ([Time](#)) const
discount calculation
- [Rate](#) [forwardImpl](#) ([Time](#)) const
instantaneous forward-rate calculation
- [Rate](#) [compoundForwardImpl](#) ([Time](#) t, [Integer](#) compFreq) const
compound forward-rate calculation

Friends

- class [FFObjFunction](#)

7.402.2 Constructor & Destructor Documentation

- 7.402.2.1 [PiecewiseFlatForward](#) (const [Date](#) & *today'sDate*, const [Date](#) & *referenceDate*, const std::vector< boost::shared_ptr< [RateHelper](#) > > & *instruments*, const [DayCounter](#) & *dayCounter*, [Real](#) *accuracy* = 1.0e-12)

Deprecated

use a constructor without today's date

7.402.2.2 **PiecewiseFlatForward** (const [Date](#) & *today'sDate*, const std::vector< [Date](#) > & *dates*, const std::vector< [Rate](#) > & *forwards*, const [DayCounter](#) & *dayCounter*)

Deprecated

use the constructor without today's date

7.402.2.3 **PiecewiseFlatForward** (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 $t = 0$.

7.402.3 Member Function Documentation

7.402.3.1 **void update ()** [virtual]

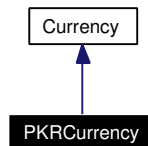
This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Reimplemented from [LazyObject](#).

7.403 PKRCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for PKRCurrency:



7.403.1 Detailed Description

Pakistani rupee.

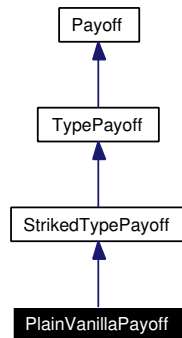
The ISO three-letter code is PKR; the numeric code is 586. It is divided in 100 paisa.

ingroup currencies

7.404 PlainVanillaPayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for PlainVanillaPayoff:



7.404.1 Detailed Description

Plain-vanilla payoff.

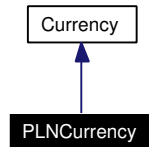
Public Member Functions

- **PlainVanillaPayoff** (Option::Type type, [Real](#) strike)
- [Real](#) **operator()** ([Real](#) price) const

7.405 PLNCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for PLNCurrency:



7.405.1 Detailed Description

Polish zloty.

The ISO three-letter code is PLN; the numeric code is 985. It is divided in 100 groszy.

ingroup currencies

7.406 PoissonDistribution Class Reference

```
#include <ql/Math/poissondistribution.hpp>
```

7.406.1 Detailed Description

Normal distribution function.

Given an integer k , it returns its probability in a Poisson distribution.

Tests

the correctness of the returned value is tested by checking it against known good results.

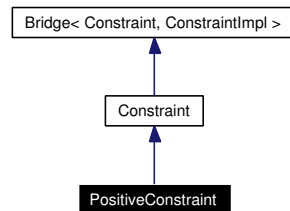
Public Member Functions

- **PoissonDistribution** ([Real](#) mu)
- **[Real](#) operator()** (BigNatural k) const

7.407 PositiveConstraint Class Reference

```
#include <ql/Optimization/constraint.hpp>
```

Inheritance diagram for PositiveConstraint:



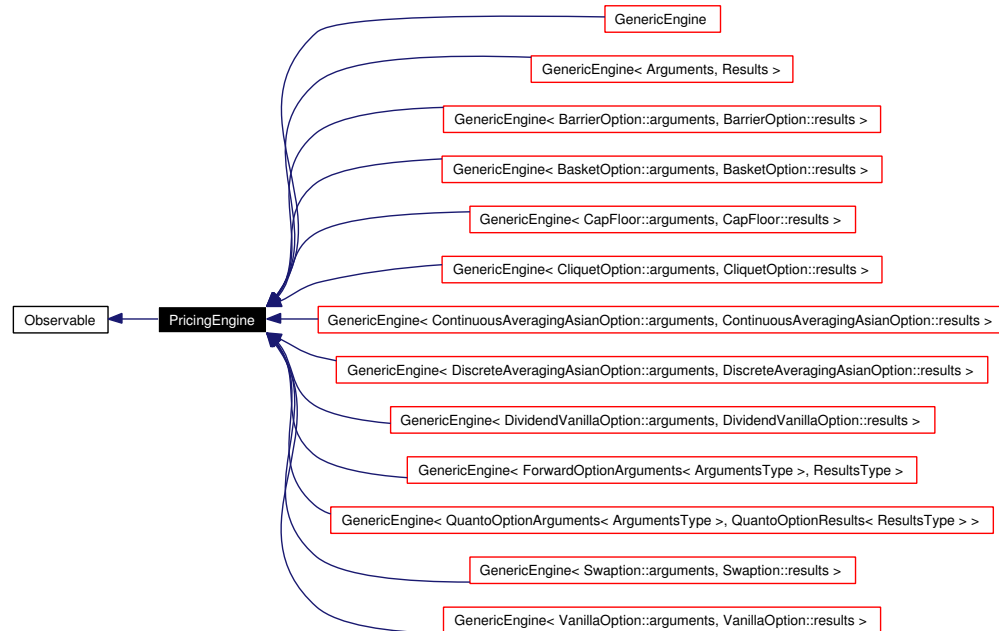
7.407.1 Detailed Description

Constraint imposing positivity to all arguments

7.408 PricingEngine Class Reference

```
#include <ql/pricingengine.hpp>
```

Inheritance diagram for PricingEngine:



7.408.1 Detailed Description

interface for pricing engines

Public Member Functions

- virtual **Arguments** * **arguments** () const =0
- virtual const **Results** * **results** () const =0
- virtual void **reset** () const =0
- virtual void **calculate** () const =0

7.409 PrimeNumbers Class Reference

```
#include <ql/Math/primenumbers.hpp>
```

7.409.1 Detailed Description

Prime numbers calculator.

Taken from "Monte Carlo Methods in Finance", by Peter Jäckel

Static Public Member Functions

- `BigNatural` [get](#) ([Size](#) absoluteIndex)
Get and store one after another.

7.410 Problem Class Reference

```
#include <ql/Optimization/problem.hpp>
```

7.410.1 Detailed Description

Constrained optimization problem.

Public Member Functions

- [Problem](#) ([CostFunction](#) &f, [Constraint](#) &c, [OptimizationMethod](#) &meth)
default constructor
- [Real value](#) (const [Array](#) &x) const
call cost function computation and increment evaluation counter
- void [gradient](#) ([Array](#) &grad_f, const [Array](#) &x) const
call cost function gradient computation and increment
- [Real valueAndGradient](#) ([Array](#) &grad_f, const [Array](#) &x) const
call cost function computation and it gradient
- [OptimizationMethod](#) & [method](#) () const
Constrained optimization method.
- [Constraint](#) & [constraint](#) () const
Constraint.
- [CostFunction](#) & [costFunction](#) () const
Cost function.
- void [minimize](#) () const
Minimization.
- [Array](#) & [minimumValue](#) () const

Protected Attributes

- [CostFunction](#) & [costFunction_](#)
Unconstrained cost function.
- [Constraint](#) & [constraint_](#)
Constraint.
- [OptimizationMethod](#) & [method_](#)
constrained optimization method

7.411 processing_iterator Class Template Reference

```
#include <ql/Utilities/processingiterator.hpp>
```

7.411.1 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/>)

Deprecated

use `boost::transform_iterator` instead

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[] (difference_type) 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.)

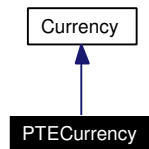
- `processing_iterator`< Iterator, UnaryFunction > `make_processing_iterator` (Iterator it, UnaryFunction p)

helper function to create processing iterators

7.412 PTECurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for PTECurrency:



7.412.1 Detailed Description

Portuguese escudo.

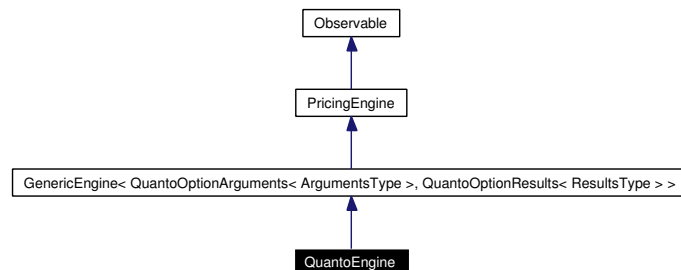
The ISO three-letter code is PTE; the numeric code is 620. It is divided in 100 centavos.

ingroup currencies

7.413 QuantoEngine Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

Inheritance diagram for QuantoEngine:



7.413.1 Detailed Description

```
template<class ArgumentsType, class ResultsType> class QuantLib::QuantoEngine<
ArgumentsType, ResultsType >
```

Quanto engine base class.

Warning:

for the time being, this engine will only work with simple Black-Scholes processes (i.e., no Merton.)

Tests

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

Public Member Functions

- **QuantoEngine** (const boost::shared_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > &)
- void **calculate** () const
- ArgumentsType * **underlyingArgs** () const

Protected Attributes

- boost::shared_ptr< [GenericEngine](#)< ArgumentsType, ResultsType > > **originalEngine_**
- ArgumentsType * **originalArguments_**
- const ResultsType * **originalResults_**

7.413.2 Member Function Documentation

7.413.2.1 ArgumentsType* underlyingArgs () const

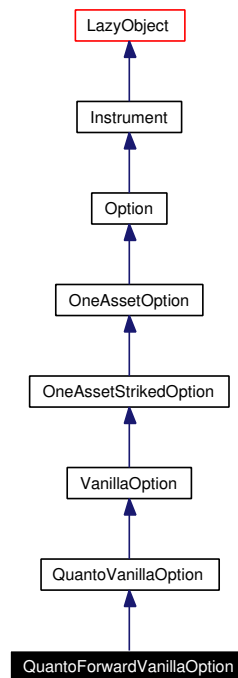
Access to the arguments of the underlying engine is needed as this engine is not able to set them completely. When necessary, it must be done by the instrument: see [QuantoForwardVanilla](#)

Option for an example.

7.414 QuantoForwardVanillaOption Class Reference

```
#include <ql/Instruments/quantoforwardvanillaoption.hpp>
```

Inheritance diagram for QuantoForwardVanillaOption:



7.414.1 Detailed Description

Quanto version of a forward vanilla option.

Public Types

- typedef [QuantoOptionArguments](#)< [ForwardVanillaOption::arguments](#) > **arguments**
- typedef [QuantoOptionResults](#)< [ForwardVanillaOption::results](#) > **results**
- typedef [QuantoEngine](#)< [ForwardVanillaOption::arguments](#), [ForwardVanillaOption::results](#) > **engine**

Public Member Functions

- **QuantoForwardVanillaOption** (const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &exchRateVolTS, const [Handle](#)< [Quote](#) > &correlation, [Real](#) moneyness, [Date](#) resetDate, const boost::shared_ptr< [BlackScholesProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine)
- void [setupArguments](#) ([Arguments](#) *) const

7.414.2 Member Function Documentation

7.414.2.1 void setupArguments ([Arguments](#) *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [QuantoVanillaOption](#).

7.415 QuantoOptionArguments Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

7.415.1 Detailed Description

```
template<class ArgumentsType> class QuantLib::QuantoOptionArguments< ArgumentsType  
>
```

Arguments for quanto option calculation

Public Member Functions

- `void validate () const`

Public Attributes

- [Real](#) `correlation`
- [Handle](#)< [YieldTermStructure](#) > `foreignRiskFreeTS`
- [Handle](#)< [BlackVolTermStructure](#) > `exchRateVolTS`

7.416 QuantoOptionResults Class Template Reference

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

7.416.1 Detailed Description

```
template<class ResultsType> class QuantLib::QuantoOptionResults< ResultsType >
```

Results from quanto option calculation

Public Member Functions

- void reset ()

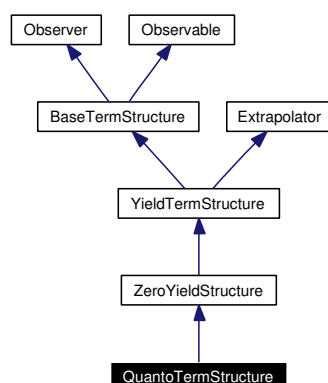
Public Attributes

- [Real](#) qvega
- [Real](#) qrho
- [Real](#) qlambda

7.417 QuantoTermStructure Class Reference

```
#include <ql/TermStructures/quantotermstructure.hpp>
```

Inheritance diagram for QuantoTermStructure:



7.417.1 Detailed Description

Quanto term structure.

Quanto term structure for modelling quanto effect in option pricing.

Note:

This term structure will remain linked to the original structures, i.e., any changes in the latters will be reflected in this structure as well.

Public Member Functions

- **QuantoTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &underlyingDividendTS, const [Handle](#)< [YieldTermStructure](#) > &riskFreeTS, const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &underlyingBlackVolTS, [Real](#) strike, const [Handle](#)< [BlackVolTermStructure](#) > &exchRateBlackVolTS, [Real](#) exchRate-ATMlevel, [Real](#) underlyingExchRateCorrelation)

YieldTermStructure interface

- [DayCounter](#) [dayCounter](#) () const
the day counter used for date/time conversion
- [Calendar](#) [calendar](#) () const
the calendar used for reference date calculation
- const [Date](#) & [todaysDate](#) () const
today's date
- const [Date](#) & [referenceDate](#) () const
the reference date, i.e., the date at which discount = 1

- [Date maxDate](#) () const
the latest date for which the curve can return rates

Protected Member Functions

- [Rate zeroYieldImpl](#) (Time) const
returns the zero yield as seen from the evaluation date

7.417.2 Member Function Documentation

7.417.2.1 const [Date](#) & todaysDate () const [virtual]

today's date

Deprecated

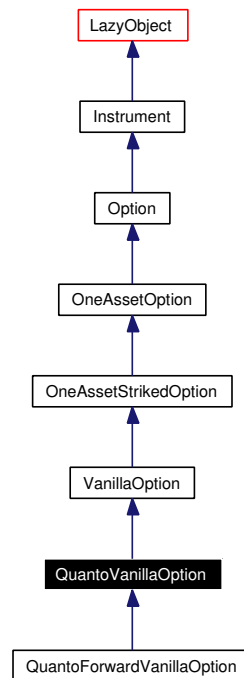
use [Settings::instance\(\).evaluationDate\(\)](#).

Reimplemented from [BaseTermStructure](#).

7.418 QuantoVanillaOption Class Reference

```
#include <ql/Instruments/quantovanillaoption.hpp>
```

Inheritance diagram for QuantoVanillaOption:



7.418.1 Detailed Description

quanto version of a vanilla option

Public Types

- typedef [QuantoOptionArguments](#)< VanillaOption::arguments > **arguments**
- typedef [QuantoOptionResults](#)< VanillaOption::results > **results**
- typedef [QuantoEngine](#)< VanillaOption::arguments, VanillaOption::results > **engine**

Public Member Functions

- **QuantoVanillaOption** (const [Handle](#)< [YieldTermStructure](#) > &foreignRiskFreeTS, const [Handle](#)< [BlackVolTermStructure](#) > &exchRateVolTS, const [Handle](#)< [Quote](#) > &correlation, const boost::shared_ptr< [BlackScholesProcess](#) > &stochProc, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine)
- void [setupArguments](#) ([Arguments](#) *) const

greeks

- [Real](#) [qvega](#) () const

- [Real](#) `qrho` () const
- [Real](#) `qlambda` () const

Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

Protected Attributes

- [Handle](#)< [YieldTermStructure](#) > `foreignRiskFreeTS_`
- [Handle](#)< [BlackVolTermStructure](#) > `exchRateVolTS_`
- [Handle](#)< [Quote](#) > `correlation_`
- [Real](#) `qvega_`
- [Real](#) `qrho_`
- [Real](#) `qlambda_`

7.418.2 Member Function Documentation

7.418.2.1 void [setupArguments](#) ([Arguments](#) *) const [virtual]

When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [OneAssetStrikedOption](#).

Reimplemented in [QuantoForwardVanillaOption](#).

7.418.2.2 void [setupExpired](#) () const [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [OneAssetOption](#).

7.418.2.3 void [performCalculations](#) () const [protected, virtual]

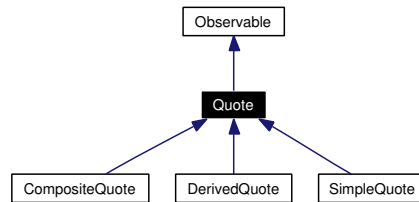
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [OneAssetStrikedOption](#).

7.419 Quote Class Reference

```
#include <ql/quote.hpp>
```

Inheritance diagram for Quote:



7.419.1 Detailed Description

purely virtual base class for market observables

Tests

the observability of class instances is tested.

Public Member Functions

- virtual [Real value](#) () const =0
returns the current value

7.420 RamdomizedLDS Class Template Reference

```
#include <ql/RandomNumbers/randomizedlds.hpp>
```

7.420.1 Detailed Description

```
template<class LDS, class PRS = RandomSequenceGenerator<MersenneTwisterUniformRng>> class QuantLib::RamdomizedLDS< LDS, PRS >
```

Randomized (random shift) low-discrepancy sequence.

Random-shifts a uniform low-discrepancy sequence of dimension N by adding (modulo 1 for each coordinate) a pseudo-random uniform deviate in $(0, 1)^N$. It is used for implementing Randomized Quasi Monte Carlo.

The uniform low discrepancy sequence is supplied by LDS; the uniform pseudo-random sequence is supplied by PRS.

Both class LDS and PRS must implement the following interface:

```
LDS::sample_type LDS::nextSequence() const;
Size LDS::dimension() const;
```

Precondition:

LDS and PRS must have the same dimension N

Warning:

Inverting LDS and PRS is possible, but it doesn't make sense

Todo

implement the other randomization algorithms

Tests

correct initialization is tested.

Public Types

- typedef [Sample](#)< [Array](#) > **sample_type**

Public Member Functions

- **RamdomizedLDS** (const LDS &lds, const PRS &prsg)
- **RamdomizedLDS** (const LDS &lds)
- **RamdomizedLDS** ([Size](#) dimensionality, [BigNatural](#) ldsSeed=0, [BigNatural](#) prsSeed=0)
- const [sample_type](#) & **nextSequence** () const
returns next sample using a given randomizing vector
- const [sample_type](#) & **lastSequence** () const
- void **nextRandomizer** ()
- [Size](#) **dimension** () const

7.420.2 Member Function Documentation

7.420.2.1 void nextRandomizer ()

update the randomizing vector and re-initialize the low discrepancy generator

7.421 RandomSequenceGenerator Class Template Reference

```
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
```

7.421.1 Detailed Description

template<class RNG> class QuantLib::RandomSequenceGenerator< RNG >

Random sequence generator based on a pseudo-random number generator.

Random sequence generator based on a pseudo-random number generator RNG.

Class RNG must implement the following interface:

```
RNG::sample_type RNG::next() const;
```

Warning:

do not use with low-discrepancy sequence generator

Public Types

- typedef [Sample](#)< [Array](#) > **sample_type**

Public Member Functions

- **RandomSequenceGenerator** ([Size](#) dimensionality, const RNG &rng)
- **RandomSequenceGenerator** ([Size](#) dimensionality, BigNatural seed=0)
- const [sample_type](#) & **nextSequence** () const
- std::vector< BigNatural > **nextInt32Sequence** () const
- const [sample_type](#) & **lastSequence** () const
- [Size](#) **dimension** () const

7.422 RateFormatter Class Reference

```
#include <ql/basicdataformatters.hpp>
```

7.422.1 Detailed Description

Formats rates for output.

Formatting is in percentage form (xx.xxxxx)

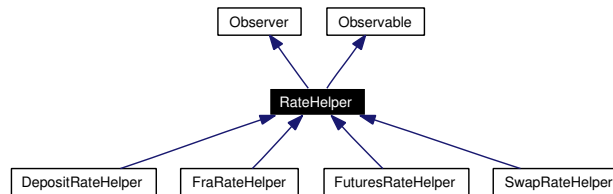
Static Public Member Functions

- `std::string toString` ([Rate](#) rate, [Integer](#) precision=5)

7.423 RateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for RateHelper:



7.423.1 Detailed Description

Base class for rate helpers.

This class provides an abstraction for the instruments used to bootstrap a term structure. It is advised that a rate helper for an instrument contains an instance of the actual instrument class to ensure consistency between the algorithms used during bootstrapping and later instrument pricing. This is not yet fully enforced in the available rate helpers, though - only [SwapRateHelper](#) contains a [Swap](#) instrument for the time being.

Public Member Functions

- [RateHelper](#) (const [Handle](#)< [Quote](#) > "e)
- [RateHelper](#) ([Real](#) quote)

RateHelper interface

- [Real](#) [quoteError](#) () const
- [Real](#) [referenceQuote](#) () const
- virtual [Real](#) [impliedQuote](#) () const =0
- virtual [DiscountFactor](#) [discountGuess](#) () const
- virtual void [setTermStructure](#) ([YieldTermStructure](#) *)
sets the term structure to be used for pricing
- virtual [Date](#) [latestDate](#) () const =0
latest relevant date
- [Date](#) [maturity](#) () const
maturity date

Observer interface

- void [update](#) ()

Protected Attributes

- [Handle](#)< [Quote](#) > quote_
- [YieldTermStructure](#) * termStructure_

7.423.2 Member Function Documentation

7.423.2.1 virtual void setTermStructure ([YieldTermStructure](#) *) [virtual]

sets the term structure to be used for pricing

Warning:

Being a pointer and not a `shared_ptr`, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented in [DepositRateHelper](#), [FraRateHelper](#), and [SwapRateHelper](#).

7.423.2.2 virtual [Date](#) latestDate () const [pure virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implemented in [DepositRateHelper](#), [FraRateHelper](#), [FuturesRateHelper](#), and [SwapRateHelper](#).

7.423.2.3 [Date](#) maturity () const

maturity date

Deprecated

renamed to [latestDate\(\)](#)

7.423.2.4 void update () [virtual]

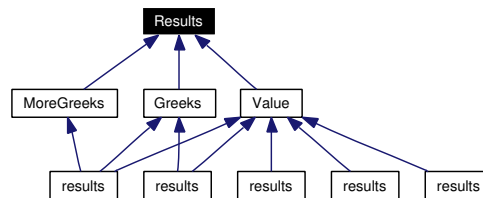
This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.424 Results Class Reference

```
#include <ql/argsandresults.hpp>
```

Inheritance diagram for Results:



7.424.1 Detailed Description

base class for generic result groups

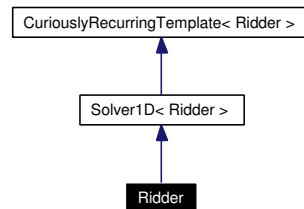
Public Member Functions

- virtual void **reset** ()=0

7.425 Ridder Class Reference

```
#include <ql/Solvers1D/ridder.hpp>
```

Inheritance diagram for Ridder:



7.425.1 Detailed Description

Ridder 1-D solver

Tests

the correctness of the returned values is tested by checking them against known good results.

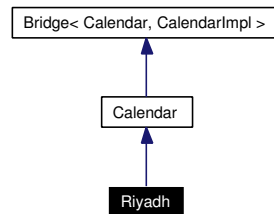
Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAcc) const`

7.426 Riyadh Class Reference

```
#include <ql/Calendars/riyadh.hpp>
```

Inheritance diagram for Riyadh:



7.426.1 Detailed Description

Riyadh calendar

Holidays:

- Fridays

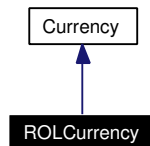
Other holidays for which no rule is given (data available for 2004-2005 only:)

- EID AL-ADHA
- EID AL-FITR

7.427 ROLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for ROLCurrency:



7.427.1 Detailed Description

Romanian leu.

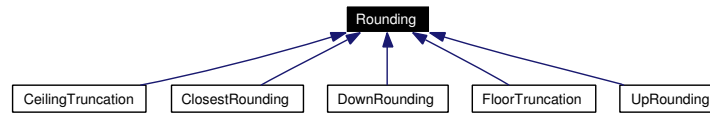
The ISO three-letter code is ROL; the numeric code is 642. It is divided in 100 bani.

ingroup currencies

7.428 Rounding Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for Rounding:



7.428.1 Detailed Description

basic rounding class

Tests

the correctness of the returned values is tested by checking them against known good results.

Inspectors

- [Integer](#) **precision** () const
- [Type](#) **type** () const
- [Integer](#) **roundingDigit** () const

Public Types

- enum [Type](#) {
[None](#), [Up](#), [Down](#), [Closest](#),
[Floor](#), [Ceiling](#) }
rounding methods

Public Member Functions

- [Rounding](#) ()
default constructor
- [Rounding](#) ([Integer](#) precision, [Type](#) type=Closest, [Integer](#) digit=5)
- [Decimal operator\(\)](#) ([Decimal](#) value) const
perform rounding

7.428.2 Member Enumeration Documentation

7.428.2.1 enum [Type](#)

rounding methods

The rounding methods follow the OMG specification available at <ftp://ftp.omg.org/pub/docs/formal/00-06-29.pdf>

Warning:

the names of the [Floor](#) and Ceiling methods might be misleading

Enumeration values:

None do not round: return the number unmodified

Up the first decimal place past the precision will be rounded up. This differs from the OMG rule which rounds up only if the decimal to be rounded is greater than or equal to the rounding digit

Down all decimal places past the precision will be truncated

Closest the first decimal place past the precision will be rounded up if greater than or equal to the rounding digit; this corresponds to the OMG round-up rule. When the rounding digit is 5, the result will be the one closest to the original number, hence the name.

Floor positive numbers will be rounded up and negative numbers will be rounded down using the OMG round up and round down rules

Ceiling positive numbers will be rounded down and negative numbers will be rounded up using the OMG round up and round down rules

7.428.3 Constructor & Destructor Documentation

7.428.3.1 [Rounding \(\)](#)

default constructor

Instances built through this constructor don't perform any rounding.

7.429 SalvagingAlgorithm Struct Reference

```
#include <ql/Math/pseudosqrt.hpp>
```

7.429.1 Detailed Description

algorithm used for matricial pseudo square root

Public Types

- enum Type { None, Spectral, Hypersphere }

7.430 Sample Struct Template Reference

```
#include <ql/MonteCarlo/sample.hpp>
```

7.430.1 Detailed Description

```
template<class T> struct QuantLib::Sample< T >
```

weighted sample

Public Types

- typedef T value_type

Public Member Functions

- Sample (const T &value, Real weight)

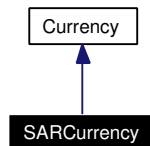
Public Attributes

- T value
- Real weight

7.431 SARCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for SARCurrency:



7.431.1 Detailed Description

Saudi riyal.

The ISO three-letter code is SAR; the numeric code is 682. It is divided in 100 halalat.

ingroup currencies

7.432 Schedule Class Reference

```
#include <ql/schedule.hpp>
```

7.432.1 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, [Frequency](#) frequency, [BusinessDayConvention](#) convention, const [Date](#) &stubDate=[Date](#)(), bool startFromEnd=false, bool longFinal=false)
- **Schedule** (const std::vector< [Date](#) > &, const [Calendar](#) &calendar, [BusinessDayConvention](#) convention)

Date access

- [Size](#) **size** () const
- const [Date](#) & **operator[]** ([Size](#) i) const
- const [Date](#) & **date** ([Size](#) i) const
- const std::vector< [Date](#) > & **dates** () const
- bool **isRegular** ([Size](#) i) const

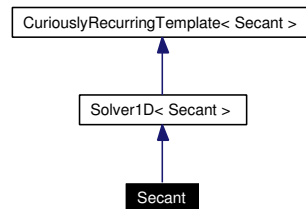
Other inspectors

- const [Calendar](#) & **calendar** () const
- const [Date](#) & **startDate** () const
- const [Date](#) & **endDate** () const
- [Frequency](#) **frequency** () const
- [BusinessDayConvention](#) **businessDayConvention** () const

7.433 Secant Class Reference

```
#include <ql/Solvers1D/secant.hpp>
```

Inheritance diagram for Secant:



7.433.1 Detailed Description

Secant 1-D solver

Tests

the correctness of the returned values is tested by checking them against known good results.

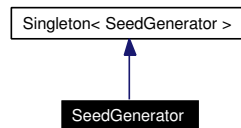
Public Member Functions

- `template<class F> Real solveImpl (const F &f, Real xAccuracy) const`

7.434 SeedGenerator Class Reference

```
#include <ql/RandomNumbers/seedgenerator.hpp>
```

Inheritance diagram for SeedGenerator:



7.434.1 Detailed Description

Random seed generator.

Random number generator used for automatic generation of initialization seeds.

Tests

correct initializaion of the single instance is tested.

Public Member Functions

- unsigned long `get ()`

Friends

- class `Singleton<SeedGenerator>`

7.435 SegmentIntegral Class Reference

```
#include <ql/Math/segmentintegral.hpp>
```

7.435.1 Detailed Description

Integral of a one-dimensional function.

Given a number N of intervals, the integral of a function f between a and b is calculated by means of the trapezoid formula

$$\int_a^b f dx = \frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \dots + f(x_{N-1}) + \frac{1}{2}f(x_N)$$

where $x_0 = a$, $x_N = b$, and $x_i = a + i\Delta x$ with $\Delta x = (b - a)/N$.

Tests

the correctness of the result is tested by checking it against known good values.

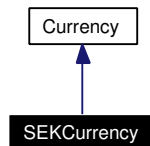
Public Member Functions

- **SegmentIntegral** ([Size](#) intervals)
- **template<class F> [Real](#) operator()** (const F &f, [Real](#) a, [Real](#) b) const

7.436 SEKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for SEKCurrency:



7.436.1 Detailed Description

Swedish krona.

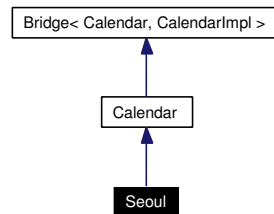
The ISO three-letter code is SEK; the numeric code is 752. It is divided in 100 öre.

ingroup currencies

7.437 Seoul Class Reference

```
#include <ql/Calendars/seoul.hpp>
```

Inheritance diagram for Seoul:



7.437.1 Detailed Description

Seoul calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Independence Day, March 1st
- Arbour Day, April 5th
- Labor Day, May 1st
- Children's Day, May 5th
- Memorial Day, June 6th
- Constitution Day, July 17th
- Liberation Day, August 15th
- National Fondation Day, October 3th
- Christmas Day, December 25th

Other holidays for which no rule is given (data available for 2004-2006 only:)

- Lunar New Year
- Election Day 2004
- Buddha's birthday
- Harvest Moon Day

Data from <http://www.kofex.com>

7.438 SequenceFormatter Class Reference

```
#include <ql/basicdataformatters.hpp>
```

7.438.1 Detailed Description

Formats numeric sequences for output.

Static Public Member Functions

- `template<class Iterator> std::string toString (Iterator begin, Iterator end, Integer precision=6, Integer digits=0, Size elementsPerRow=QL_MAX_INTEGER)`

7.439 SequenceStatistics Class Template Reference

```
#include <ql/Math/sequencestatistics.hpp>
```

7.439.1 Detailed Description

```
template<class StatisticsType = Statistics> class QuantLib::SequenceStatistics< StatisticsType
>
```

Statistics analysis of N-dimensional (sequence) data.

It provides 1-dimensional statistics as discrepancy plus N-dimensional (sequence) statistics (e.g. mean, variance, skewness, kurtosis, etc.) with one component for each dimension of the sample space.

For most of the statistics this class relies on the StatisticsType underlying class to provide 1-D methods that will be iterated for all the components of the N-D data. These lifted methods are the union of all the methods that might be requested to the 1-D underlying StatisticsType class, with the usual compile-time checks provided by the template approach.

Tests

the correctness of the returned values is tested by checking them against numerical calculations.

Public Types

- typedef StatisticsType **statistics_type**

Public Member Functions

- **SequenceStatistics** ([Size](#) dimension)

inspectors

- [Size](#) **size** () const

covariance and correlation

- [Disposable](#)< [Matrix](#) > **covariance** () const
returns the covariance [Matrix](#)
- [Disposable](#)< [Matrix](#) > **correlation** () const
returns the correlation [Matrix](#)

1-D inspectors lifted from underlying statistics class

- [Size](#) **samples** () const
- [Real](#) **weightSum** () const

N-D inspectors lifted from underlying statistics class

- `std::vector< Real > mean () const`
- `std::vector< Real > variance () const`
- `std::vector< Real > standardDeviation () const`
- `std::vector< Real > downsideVariance () const`
- `std::vector< Real > downsideDeviation () const`
- `std::vector< Real > semiVariance () const`
- `std::vector< Real > semiDeviation () const`
- `std::vector< Real > errorEstimate () const`
- `std::vector< Real > skewness () const`
- `std::vector< Real > kurtosis () const`
- `std::vector< Real > min () const`
- `std::vector< Real > max () const`
- `std::vector< Real > gaussianPercentile (Real y) const`
- `std::vector< Real > percentile (Real y) const`
- `std::vector< Real > gaussianPotentialUpside (Real percentile) const`
- `std::vector< Real > potentialUpside (Real percentile) const`
- `std::vector< Real > gaussianValueAtRisk (Real percentile) const`
- `std::vector< Real > valueAtRisk (Real percentile) const`
- `std::vector< Real > gaussianExpectedShortfall (Real percentile) const`
- `std::vector< Real > expectedShortfall (Real percentile) const`
- `std::vector< Real > regret (Real target) const`
- `std::vector< Real > gaussianShortfall (Real target) const`
- `std::vector< Real > shortfall (Real target) const`
- `std::vector< Real > gaussianAverageShortfall (Real target) const`
- `std::vector< Real > averageShortfall (Real target) const`

Modifiers

- `void reset (Size dimension=0)`
- `template<class Sequence> void add (const Sequence &sample, Real weight=1.0)`
- `template<class Iterator> void add (Iterator begin, Iterator end, Real weight=1.0)`

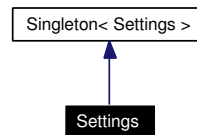
Protected Attributes

- `Size dimension_`
- `std::vector< statistics_type > stats_`
- `std::vector< Real > results_`
- `Matrix quadraticSum_`

7.440 Settings Class Reference

```
#include <ql/settings.hpp>
```

Inheritance diagram for Settings:



7.440.1 Detailed Description

global repository for run-time library settings

Public Member Functions

Evaluation date

- [Date](#) `evaluationDate ()` const
the date at which pricing is to be performed
- void `setEvaluationDate (const Date &)`
change the evaluation date and notify registered instruments
- `boost::shared_ptr<Observable> evaluationDateGuard ()` const
observable sending notification when the evaluation date changes

Friends

- class `Singleton<Settings>`

7.440.2 Member Function Documentation

7.440.2.1 [Date](#) `evaluationDate ()` const

the date at which pricing is to be performed

If not set, the current date will be used

7.440.2.2 void `setEvaluationDate (const Date &)`

change the evaluation date and notify registered instruments

Note:

settings the evaluation date to the null date will cause `evaluationDate()` to return today's date.

7.440.2.3 boost::shared_ptr< [Observable](#) > evaluationDateGuard () const

observable sending notification when the evaluation date changes

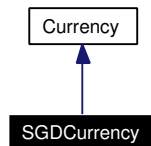
Warning:

this observable does not send a notification when the evaluation date changes for natural causes—i.e., a date was not explicitly set (which results in today's date being used for pricing) and the current date changes as the clock strikes midnight.

7.441 SGDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for SGDCurrency:



7.441.1 Detailed Description

[Singapore](#) dollar.

The ISO three-letter code is SGD; the numeric code is 702. It is divided in 100 cents.

ingroup currencies

7.442 Short Class Template Reference

```
#include <ql/CashFlows/shortindexedcoupon.hpp>
```

7.442.1 Detailed Description

```
template<class IndexedCouponType> class QuantLib::Short< IndexedCouponType >
```

Short indexed coupon

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **Short** ([Real](#) nominal, const [Date](#) &paymentDate, const boost::shared_ptr< [Xibor](#) > &index, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())
- **Real amount** () const
inhibit calculation

7.442.2 Member Function Documentation

7.442.2.1 [Real amount](#) () const

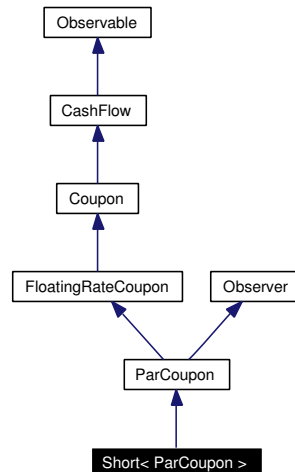
inhibit calculation

Unlike [ParCoupon](#), this coupon can't calculate its fixing for future dates, either.

7.443 Short< ParCoupon > Class Template Reference

```
#include <ql/CashFlows/shortfloatingcoupon.hpp>
```

Inheritance diagram for Short< ParCoupon >:



7.443.1 Detailed Description

```
template<> class QuantLib::Short< ParCoupon >
```

Short coupon at par on a term structure

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **Short** ([Real](#) nominal, const [Date](#) &paymentDate, const boost::shared_ptr< [Xibor](#) > &index, const [Date](#) &startDate, const [Date](#) &endDate, [Integer](#) fixingDays, [Spread](#) spread=0.0, const [Date](#) &refPeriodStart=[Date](#)(), const [Date](#) &refPeriodEnd=[Date](#)(), const [DayCounter](#) &dayCounter=[DayCounter](#)())
- **Real amount** () const
throws when an interpolated fixing is needed

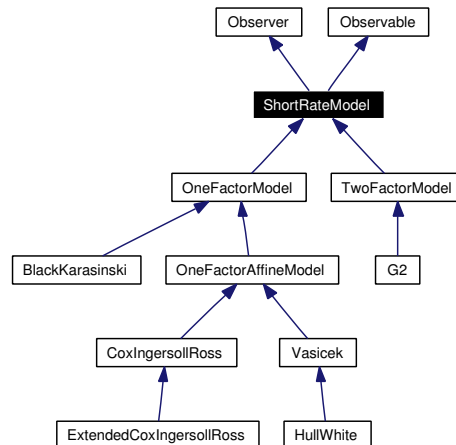
Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

7.444 ShortRateModel Class Reference

```
#include <ql/ShortRateModels/model.hpp>
```

Inheritance diagram for ShortRateModel:



7.444.1 Detailed Description

Abstract short-rate model class.

Public Member Functions

- **ShortRateModel** ([Size](#) nArguments)
- void [update](#) ()
- virtual boost::shared_ptr< [Lattice](#) > [tree](#) (const [TimeGrid](#) &) const =0
- void [calibrate](#) (const std::vector< boost::shared_ptr< [CalibrationHelper](#) > > &, [OptimizationMethod](#) &method, const [Constraint](#) &constraint=[Constraint](#)())
Calibrate to a set of market instruments (caps/swaptions).
- const boost::shared_ptr< [Constraint](#) > & [constraint](#) () const
- [Disposable](#)< [Array](#) > [params](#) () const
Returns array of arguments on which calibration is done.
- void [setParams](#) (const [Array](#) ¶ms)

Protected Member Functions

- virtual void [generateArguments](#) ()

Protected Attributes

- std::vector< [Parameter](#) > [arguments_](#)
- boost::shared_ptr< [Constraint](#) > [constraint_](#)

Friends

- class `CalibrationFunction`

7.444.2 Member Function Documentation

7.444.2.1 `void update ()` [virtual]

This method must be implemented in derived classes. An instance of `Observer` does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements [Observer](#).

7.444.2.2 `void calibrate (const std::vector< boost::shared_ptr< CalibrationHelper > > &, OptimizationMethod & method, const Constraint & constraint = Constraint\(\))`

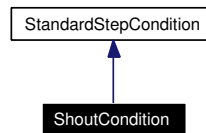
Calibrate to a set of market instruments (caps/swaptions).

An additional constraint can be passed which must be satisfied in addition to the constraints of the model.

7.445 ShoutCondition Class Reference

```
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

Inheritance diagram for ShoutCondition:



7.445.1 Detailed Description

Shout option condition.

A shout option is an option where the holder has the right to lock in a minimum value for the payoff at one (shout) time during the option's life. The minimum value is the option's intrinsic value at the shout time.

Todo

unify the intrinsicValues/Payoff thing

Public Member Functions

- **ShoutCondition** (Option::Type type, [Real](#) strike, [Time](#) resTime, [Rate](#) rate)
- **ShoutCondition** (const [Array](#) &intrinsicValues, [Time](#) resTime, [Rate](#) rate)
- void **applyTo** ([Array](#) &a, [Time](#) t) const
- void **applyTo** (boost::shared_ptr< [DiscretizedAsset](#) > asset) const

7.445.2 Member Function Documentation

7.445.2.1 void **applyTo** (boost::shared_ptr< [DiscretizedAsset](#) > *asset*) const [virtual]

Deprecated

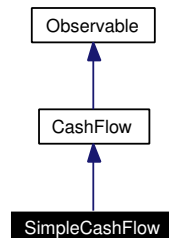
use `adjustValues()` on the asset itself

Implements [StepCondition](#).

7.446 SimpleCashFlow Class Reference

```
#include <ql/CashFlows/simplecashflow.hpp>
```

Inheritance diagram for SimpleCashFlow:



7.446.1 Detailed Description

Predetermined cash flow.

This cash flow pays a predetermined amount at a given date.

Public Member Functions

- **SimpleCashFlow** ([Real](#) amount, const [Date](#) &date)

CashFlow interface

- [Real](#) **amount** () const
returns the amount of the cash flow
- [Date](#) **date** () const
returns the date at which the cash flow is settled

Visitability

- virtual void **accept** ([AcyclicVisitor](#) &)

7.446.2 Member Function Documentation

7.446.2.1 [Real](#) amount () const [virtual]

returns the amount of the cash flow

Note:

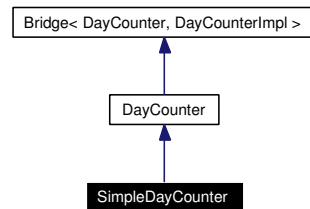
The amount is not discounted, i.e., it is the actual amount paid at the cash flow date.

Implements [CashFlow](#).

7.447 SimpleDayCounter Class Reference

```
#include <ql/DayCounters/simpliedaycounter.hpp>
```

Inheritance diagram for SimpleDayCounter:



7.447.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

This day counter tries to ensure that whole-month distances are returned as a simple fraction, i.e., 1 year = 1.0, 6 months = 0.5, 3 months = 0.25 and so forth.

Warning:

this day counter should be used together with [NullCalendar](#), which ensures that dates at whole-month distances share the same day of month. It is **not** guaranteed to work with any other calendar.

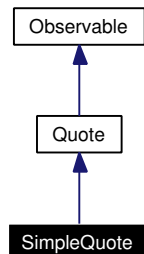
Tests

the correctness of the results is checked against known good values.

7.448 SimpleQuote Class Reference

```
#include <ql/quote.hpp>
```

Inheritance diagram for SimpleQuote:



7.448.1 Detailed Description

market element returning a stored value

Public Member Functions

- SimpleQuote ([Real](#) value)

Quote interface

- [Real value](#) () const
returns the current value

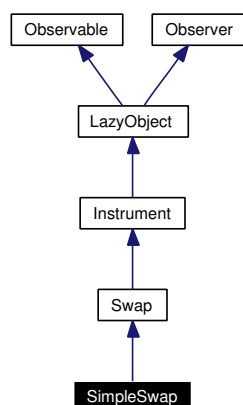
Modifiers

- void setValue ([Real](#) value)

7.449 SimpleSwap Class Reference

```
#include <ql/Instruments/simpleswap.hpp>
```

Inheritance diagram for SimpleSwap:



7.449.1 Detailed Description

Simple fixed-rate vs Libor swap.

Tests

- the correctness of the returned value is tested by checking that the price of a swap paying the fair fixed rate is null.
- the correctness of the returned value is tested by checking that the price of a swap receiving the fair floating-rate spread is null.
- the correctness of the returned value is tested by checking that the price of a swap decreases with the paid fixed rate.
- the correctness of the returned value is tested by checking that the price of a swap increases with the received floating-rate spread.
- the correctness of the returned value is tested by checking it against a known good value.

Public Member Functions

- **SimpleSwap** (bool payFixedRate, [Real](#) nominal, const [Schedule](#) &fixedSchedule, [Rate](#) fixedRate, const [DayCounter](#) &fixedDayCount, const [Schedule](#) &floatSchedule, const boost::shared_ptr< [Xibor](#) > &index, [Integer](#) indexFixingDays, [Spread](#) spread, const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- [Rate](#) fairRate () const
- [Spread](#) fairSpread () const
- [Real](#) fixedLegBPS () const
- [Real](#) floatingLegBPS () const
- [Rate](#) fixedRate () const
- [Spread](#) spread () const
- [Real](#) nominal () const
- bool payFixedRate () const
- const std::vector< boost::shared_ptr< [CashFlow](#) > > & fixedLeg () const

- `const std::vector< boost::shared_ptr< CashFlow > > & floatingLeg () const`
- `void setupArguments (Arguments *args) const`

7.449.2 Member Function Documentation

7.449.2.1 `void setupArguments (Arguments *args) const` [virtual]

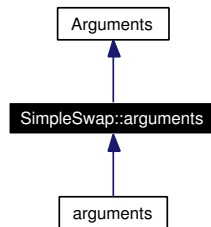
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

7.450 SimpleSwap::arguments Class Reference

```
#include <ql/Instruments/simpleswap.hpp>
```

Inheritance diagram for SimpleSwap::arguments:



7.450.1 Detailed Description

Arguments for simple swap calculation

Public Member Functions

- void **validate** () const

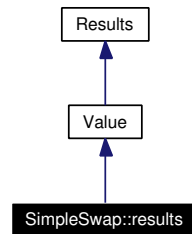
Public Attributes

- bool **payFixed**
- **Real** **nominal**
- std::vector< **Time** > **fixedResetTimes**
- std::vector< **Time** > **fixedPayTimes**
- std::vector< **Real** > **fixedCoupons**
- std::vector< **Time** > **floatingAccrualTimes**
- std::vector< **Time** > **floatingResetTimes**
- std::vector< **Time** > **floatingPayTimes**
- std::vector< **Spread** > **floatingSpreads**
- **Real** **currentFloatingCoupon**

7.451 SimpleSwap::results Class Reference

```
#include <ql/Instruments/simpleswap.hpp>
```

Inheritance diagram for SimpleSwap::results:



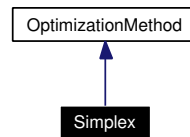
7.451.1 Detailed Description

Results from simple swap calculation

7.452 Simplex Class Reference

```
#include <ql/Optimization/simplex.hpp>
```

Inheritance diagram for Simplex:



7.452.1 Detailed Description

Multi-dimensional simplex class.

Public Member Functions

- [Simplex](#) ([Real](#) lambda, [Real](#) tol)
- virtual void [minimize](#) (const [Problem](#) &P) const
minimize the optimization problem P

7.452.2 Constructor & Destructor Documentation

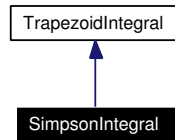
7.452.2.1 [Simplex](#) ([Real](#) lambda, [Real](#) tol)

Constructor taking as input the characteristic length and tolerance

7.453 SimpsonIntegral Class Reference

```
#include <ql/Math/simpsonintegral.hpp>
```

Inheritance diagram for SimpsonIntegral:



7.453.1 Detailed Description

Integral of a one-dimensional function.

Tests

the correctness of the result is tested by checking it against known good values.

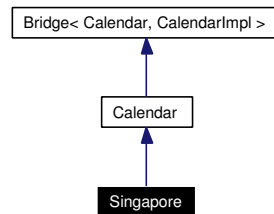
Public Member Functions

- **SimpsonIntegral** ([Real](#) accuracy, [Size](#) maxIterations=[Null](#)< [Size](#) >())
- **template<class F> [Real](#) operator()** (const F &f, [Real](#) a, [Real](#) b) const

7.454 Singapore Class Reference

```
#include <ql/Calendars/singapore.hpp>
```

Inheritance diagram for Singapore:



7.454.1 Detailed Description

Singapore calendar

Holidays:

- Saturdays
- Sundays
- New Year's day, January 1st
- Good Friday
- Labour Day, May 1st
- National Day, August 9th
- Christmas, December 25th
- Boxing Day, December 26th

Other holidays for which no rule is given (data available for 2004-2005 only:)

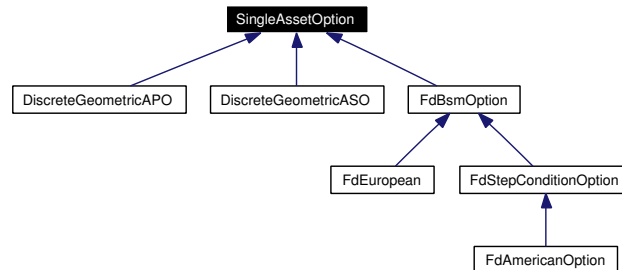
- Chinese New Year
- Hari Raya Haji
- Vesak Poya Day
- Deepavali
- Diwali
- Hari Raya Puasa

Data from <http://www.asx.com.au> and <http://www.ses.com.sg>

7.455 SingleAssetOption Class Reference

```
#include <ql/Pricers/singleassetoption.hpp>
```

Inheritance diagram for SingleAssetOption:



7.455.1 Detailed Description

Black-Scholes-Merton option.

Public Member Functions

- **SingleAssetOption** (Option::Type type, [Real](#) underlying, [Real](#) strike, [Spread](#) dividendYield, [Rate](#) riskFreeRate, [Time](#) residualTime, [Volatility](#) volatility)
- virtual void **setVolatility** ([Volatility](#) newVolatility)
- virtual void **setRiskFreeRate** ([Rate](#) newRate)
- virtual void **setDividendYield** ([Rate](#) newDividendYield)
- virtual [Real](#) **value** () const =0
- virtual [Real](#) **delta** () const =0
- virtual [Real](#) **gamma** () const =0
- virtual [Real](#) **theta** () const
- virtual [Real](#) **vega** () const
- virtual [Real](#) **rho** () const
- virtual [Real](#) **dividendRho** () const
- [Volatility](#) **impliedVolatility** ([Real](#) targetValue, [Real](#) accuracy=1e-4, [Size](#) maxEvaluations=100, [Volatility](#) minVol=QL_MIN_VOLATILITY, [Volatility](#) maxVol=QL_MAX_VOLATILITY) const
- [Spread](#) **impliedDivYield** ([Real](#) targetValue, [Real](#) accuracy=1e-4, [Size](#) maxEvaluations=100, [Spread](#) minYield=QL_MIN_DIVYIELD, [Spread](#) maxYield=QL_MAX_DIVYIELD) const
- virtual boost::shared_ptr< [SingleAssetOption](#) > **clone** () const =0

Protected Attributes

- [Real](#) **underlying_**
- [PlainVanillaPayoff](#) **payoff_**
- [Spread](#) **dividendYield_**
- [Rate](#) **riskFreeRate_**
- [Time](#) **residualTime_**
- [Volatility](#) **volatility_**

- bool `hasBeenCalculated_`
- [Real](#) `rho_`
- [Real](#) `dividendRho_`
- [Real](#) `vega_`
- [Real](#) `theta_`
- bool `rhoComputed_`
- bool `dividendRhoComputed_`
- bool `vegaComputed_`
- bool `thetaComputed_`

Static Protected Attributes

- const [Real](#) `dVolMultiplier_`
- const [Real](#) `dRMultiplier_`

Friends

- class `VolatilityFunction`
- class `DivYieldFunction`

7.455.2 Member Function Documentation

7.455.2.1 [Volatility](#) `impliedVolatility(Real targetValue, Real accuracy = 1e-4, Size maxEvaluations = 100, Volatility minVol = QL_MIN_VOLATILITY, Volatility maxVol = QL_MAX_VOLATILITY) const`

Warning:

Options with a gamma that changes sign have values that are **not** monotonic in the volatility, e.g binary options. In these cases `impliedVolatility` can fail and in any case is meaningless. Another possible source of failure is to have a `targetValue` that is not attainable with any volatility, e.g. a `targetValue` lower than the intrinsic value in the case of American options.

7.456 Singleton Class Template Reference

```
#include <ql/Patterns/singleton.hpp>
```

7.456.1 Detailed Description

```
template<class T> class QuantLib::Singleton< T >
```

Basic support for the singleton pattern.

The typical use of this class is:

```
class Foo : public Singleton<Foo> {  
    friend class Singleton<Foo>;  
private:  
    Foo() {}  
public:  
    ...  
};
```

which, albeit sub-optimal, frees one from the concerns of creating and managing the unique instance and can serve later as a single implementation point should synchronization features be added.

If the derived class provides an `initialize()` method, it will be called after creation of the unique instance.

Static Public Member Functions

- `T & instance ()`
access to the unique instance

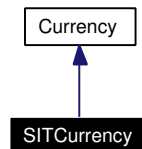
Protected Member Functions

- `void initialize ()`

7.457 SITCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for SITCurrency:



7.457.1 Detailed Description

Slovenian tolar.

The ISO three-letter code is SIT; the numeric code is 705. It is divided in 100 stotinov.

ingroup currencies

7.458 SizeFormatter Class Reference

```
#include <ql/basicdataformatters.hpp>
```

7.458.1 Detailed Description

Formats unsigned integers for output.

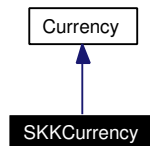
Static Public Member Functions

- `std::string toString` ([Size](#) l, [Integer](#) digits=0)
- `std::string toOrdinal` ([Size](#) l)
- `std::string toPowerOfTwo` ([Size](#) l, [Integer](#) digits=0)

7.459 SKKCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for SKKCurrency:



7.459.1 Detailed Description

Slovak koruna.

The ISO three-letter code is SKK; the numeric code is 703. It is divided in 100 halierov.

ingroup currencies

7.460 SobolRsg Class Reference

```
#include <ql/RandomNumbers/sobolrsg.hpp>
```

7.460.1 Detailed Description

Sobol low-discrepancy sequence generator.

A Gray code counter and bitwise operations are used for very fast sequence generation.

The implementation relies on primitive polynomials modulo two from the book "Monte Carlo Methods in Finance" by Peter Jäckel.

21 200 primitive polynomials modulo two are provided by default in QuantLib. Jäckel has calculated 8 129 334 polynomials, also available in a different file that can be downloaded from <http://quantlib.org>. If you need that many dimensions you must replace the default version of the `primitivpolynomials.c` file with the extended one.

The choice of initialization numbers (also know as free direction integers) is crucial for the homogeneity properties of the sequence. Sobol defines two homogeneity properties: Property A and Property A'.

The unit initialization numbers suggested in "Numerical Recipes in C", 2nd edition, by Press, Teukolsky, Vetterling, and Flannery (section 7.7) fail the test for Property A even for low dimensions.

Bratley and Fox published coefficients of the free direction integers up to dimension 40, crediting unpublished work of Sobol' and Levitan. See Bratley, P., Fox, B.L. (1988) "Algorithm 659: Implementing Sobol's quasirandom sequence generator," ACM Transactions on Mathematical Software 14:88-100. These values satisfy Property A for $d \leq 20$ and $d = 23, 31, 33, 34, 37$; Property A' holds for $d \leq 6$.

Jäckel provides in his book (section 8.3) initialization numbers up to dimension 32. Coefficients for $d \leq 8$ are the same as in Bradley-Fox, so Property A' holds for $d \leq 6$ but Property A holds for $d \leq 32$.

The implementation of Lemieux, Cieslak, and Luttmmer includes coefficients of the free direction integers up to dimension 360. Coefficients for $d \leq 40$ are the same as in Bradley-Fox. For dimension $40 < d \leq 360$ the coefficients have been calculated as optimal values based on the "resolution" criterion. See "RandQMC user's guide - A package for randomized quasi-Monte Carlo methods in C," by C. Lemieux, M. Cieslak, and K. Luttmmer, version January 13 2004, and references cited there (<http://www.math.ualgary.ca/~lemieux/randqmc.html>). The values up to $d \leq 360$ has been provided to the QuantLib team by Christiane Lemieux, private communication, September 2004.

For more info on Sobol' sequences see also "Monte Carlo Methods in Financial Engineering," by P. Glasserman, 2004, Springer, section 5.2.3

Tests

- a) the correctness of the returned values is tested by reproducing known good values.
- b) the correctness of the returned values is tested by checking their discrepancy against known good values.

Public Types

- typedef [Sample](#)< [Array](#) > `sample_type`

- enum `DirectionIntegers` { `Unit`, `Jaeckel`, `SobolLevitan`, `SobolLevitanLemieux` }

Public Member Functions

- `SobolRsg` (`Size` dimensionality, unsigned long seed=0, `DirectionIntegers` directionIntegers=`Jaeckel`)
- const std::vector< unsigned long > & `nextInt32Sequence` () const
- const `SobolRsg::sample_type` & `SobolRsg::nextSequence` () const
- const `sample_type` & `lastSequence` () const
- `Size` `dimension` () const

7.460.2 Constructor & Destructor Documentation

- 7.460.2.1 `SobolRsg` (`Size` dimensionality, unsigned long seed = 0, `DirectionIntegers` directionIntegers = `Jaeckel`)

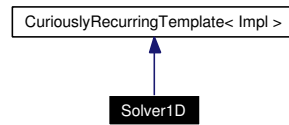
Precondition:

dimensionality must be <= PPMT_MAX_DIM

7.461 Solver1D Class Template Reference

```
#include <ql/solver1d.hpp>
```

Inheritance diagram for Solver1D:



7.461.1 Detailed Description

template<class Impl> class QuantLib::Solver1D< Impl >

Base class for 1-D solvers.

The implementation of this class uses the so-called "Barton-Nackman trick", also known as "the curiously recurring template pattern". Concrete solvers will be declared as:

```

class Foo : public Solver1D<Foo> {
public:
    ...
    template <class F>
    Real solveImpl(const F& f, Real accuracy) const {
        ...
    }
};

```

Before calling `solveImpl`, the base class will set its protected data members so that:

- `xMin_` and `xMax_` form a valid bracket;
- `fxMin_` and `fxMax_` contain the values of the function in `xMin_` and `xMax_`;
- `root_` is a valid initial guess. The implementation of `solveImpl` can safely assume all of the above.

Todo

clean up the interface so that it is clear whether the accuracy is specified for x or $f(x)$.
add target value (now the target value is 0.0)

Public Member Functions

Modifiers

- `template<class F> Real solve` (const F &f, Real accuracy, Real guess, Real step) const
- `template<class F> Real solve` (const F &f, Real accuracy, Real guess, Real xMin, Real xMax) const
- void `setMaxEvaluations` (Size evaluations)
- void `setLowerBound` (Real lowerBound)
sets the lower bound for the function domain
- void `setUpperBound` (Real upperBound)
sets the upper bound for the function domain

Protected Attributes

- [Real](#) root_
- [Real](#) xMin_
- [Real](#) xMax_
- [Real](#) fxMin_
- [Real](#) fxMax_
- [Size](#) maxEvaluations_
- [Size](#) evaluationNumber_

7.461.2 Member Function Documentation

7.461.2.1 [Real](#) solve (const F & *f*, [Real](#) *accuracy*, [Real](#) *guess*, [Real](#) *step*) const

This method returns the zero of the function f , determined with the given accuracy (i.e., x is considered a zero if $|f(x)| < accuracy$). This method contains a bracketing routine to which an initial guess must be supplied as well as a step used to scan the range of the possible bracketing values.

7.461.2.2 [Real](#) solve (const F & *f*, [Real](#) *accuracy*, [Real](#) *guess*, [Real](#) *xMin*, [Real](#) *xMax*) const

This method returns the zero of the function f , determined with the given accuracy (i.e., x is considered a zero if $|f(x)| < accuracy$). An initial guess must be supplied, as well as two values x_{\min} and x_{\max} which must bracket the zero (i.e., either $f(x_{\min}) \leq 0 \leq f(x_{\max})$, or $f(x_{\max}) \leq 0 \leq f(x_{\min})$ must be true).

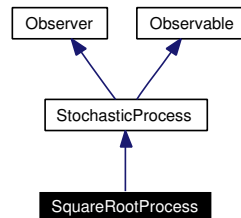
7.461.2.3 void setMaxEvaluations ([Size](#) *evaluations*)

This method sets the maximum number of function evaluations for the bracketing routine. An error is thrown if a bracket is not found after this number of evaluations.

7.462 SquareRootProcess Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for SquareRootProcess:



7.462.1 Detailed Description

Square-root process class.

This class describes a square-root process governed by

$$dx = a(b - x_t)dt + \sigma \sqrt{x_t}dW_t.$$

Public Member Functions

- **SquareRootProcess** ([Real](#) b, [Real](#) a, [Volatility](#) sigma, [Real](#) x0=0.0, const boost::shared_ptr<[StochasticProcess::discretization](#)> &d=boost::shared_ptr<[StochasticProcess::discretization](#)>(new [EulerDiscretization](#)))

StochasticProcess interface

- [Real](#) **x0** () const
returns the initial value of the state variable
- [Real](#) **drift** ([Time](#) t, [Real](#) x) const
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- [Real](#) **diffusion** ([Time](#) t, [Real](#) x) const
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$

7.463 StatsHolder Class Reference

```
#include <ql/Math/gaussianstatistics.hpp>
```

7.463.1 Detailed Description

Helper class for precomputed distributions.

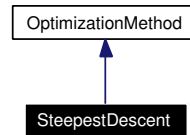
Public Member Functions

- **StatsHolder** ([Real](#) mean, [Real](#) standardDeviation)
- [Real](#) **mean** () const
- [Real](#) **standardDeviation** () const

7.464 SteepestDescent Class Reference

```
#include <ql/Optimization/steepestdescent.hpp>
```

Inheritance diagram for SteepestDescent:



7.464.1 Detailed Description

Multi-dimensional steepest-descent class.

User has to provide line-search method and optimization end criteria

search direction = $-f'(x)$

Public Member Functions

- [SteepestDescent](#) ()
default default constructor (msvc bug)
- [SteepestDescent](#) (const boost::shared_ptr< [LineSearch](#) > &lineSearch)
default constructor
- virtual [~SteepestDescent](#) ()
destructor
- virtual void [minimize](#) (const [Problem](#) &P) const
minimize the optimization problem P

7.465 `step_iterator` Class Template Reference

```
#include <ql/Utilities/steppingiterator.hpp>
```

7.465.1 Detailed Description

template<class Iterator> class QuantLib::step_iterator< Iterator >

Iterator advancing in constant steps.

This iterator advances an underlying random-access iterator in steps of n positions, where n is a positive integer given upon construction.

Public Member Functions

- **step_iterator** (const Iterator &base, [Size](#) step)
- template<class OtherIterator> **step_iterator** (const [step_iterator](#)< OtherIterator > &i, typename boost::enable_if_convertible< OtherIterator, Iterator >::type *=0)
- [Size](#) **step** () const
- void **increment** ()
- void **decrement** ()
- void **advance** (typename super_t::difference_type n)
- super_t::difference_type **distance_to** (const [step_iterator](#) &i) const

Related Functions

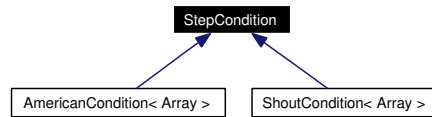
(Note that these are not member functions.)

- [step_iterator](#)< Iterator > [make_step_iterator](#) (Iterator it, [Size](#) step)
helper function to create step iterators

7.466 StepCondition Class Template Reference

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

Inheritance diagram for StepCondition:



7.466.1 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](#) (boost::shared_ptr< [DiscretizedAsset](#) >) const =0

7.466.2 Member Function Documentation

7.466.2.1 virtual void **applyTo** (boost::shared_ptr< [DiscretizedAsset](#) >) const [pure virtual]

[Deprecated](#)

use `adjustValues()` on the asset itself

Implemented in [AmericanCondition](#), and [ShoutCondition](#).

7.467 stepping_iterator Class Template Reference

```
#include <ql/Utilities/steppingiterator.hpp>
```

7.467.1 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.

Deprecated

use [step_iterator](#) instead

Public Member Functions

- `stepping_iterator` (const RandomAccessIterator &it, difference_type step)

Dereferencing

- reference `operator*` () const
- pointer `operator→` () const

Random access

- reference `operator[]` (difference_type 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

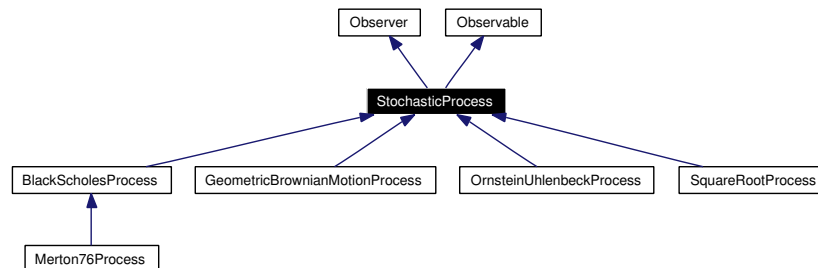
(Note that these are not member functions.)

- [stepping_iterator](#)< Iterator > [make_stepping_iterator](#) (Iterator it, typename [stepping_iterator](#)< Iterator >::difference_type step)
helper function to create stepping iterators

7.468 StochasticProcess Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for StochasticProcess:



7.468.1 Detailed Description

Stochastic 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

- virtual `Real x0 () const =0`
returns the initial value of the state variable
- virtual `Real drift (Time t, Real x) const =0`
returns the drift part of the equation, i.e. $\mu(t, x_t)$
- virtual `Real diffusion (Time t, Real x) const =0`
returns the diffusion part of the equation, i.e. $\sigma(t, x_t)$
- virtual `Real expectation (Time t0, Real x0, Time dt) const`
- virtual `Real variance (Time t0, Real x0, Time dt) const`
- virtual `Real evolve (Real change, Real currentValue) const`

Observer interface

- void `update ()`

Protected Member Functions

- `StochasticProcess (const boost::shared_ptr< discretization > &)`

Protected Attributes

- `boost::shared_ptr< discretization > discretization_`

7.468.2 Member Function Documentation

7.468.2.1 virtual **Real** expectation (**Time** *t0*, **Real** *x0*, **Time** *dt*) const [virtual]

returns the expectation $E(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [OrnsteinUhlenbeckProcess](#).

7.468.2.2 virtual **Real** variance (**Time** *t0*, **Real** *x0*, **Time** *dt*) const [virtual]

returns the variance $V(x_{t_0+\Delta t}|x_{t_0} = x_0)$ of the process after a time interval Δt according to the given discretization. This method can be overridden in derived classes which want to hard-code a particular discretization.

Reimplemented in [OrnsteinUhlenbeckProcess](#).

7.468.2.3 virtual **Real** evolve (**Real** *change*, **Real** *currentValue*) const [virtual]

applies a change to the asset value. By default; it returns $x + \Delta x$.

Reimplemented in [BlackScholesProcess](#), and [Merton76Process](#).

7.468.2.4 void update () [virtual]

This method must be implemented in derived classes. An instance of Observer does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

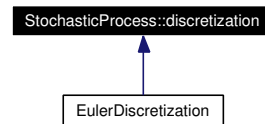
Implements [Observer](#).

Reimplemented in [BlackScholesProcess](#).

7.469 StochasticProcess::discretization Class Reference

```
#include <ql/stochasticprocess.hpp>
```

Inheritance diagram for StochasticProcess::discretization:



7.469.1 Detailed Description

discretization of a stochastic process over a given time interval

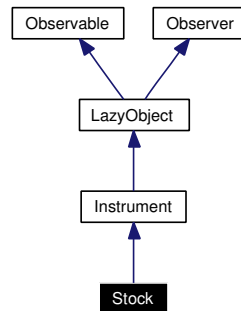
Public Member Functions

- virtual **Real** **expectation** (const **StochasticProcess** &, **Time** t0, **Real** x0, **Time** dt) const =0
- virtual **Real** **variance** (const **StochasticProcess** &, **Time** t0, **Real** x0, **Time** dt) const =0

7.470 Stock Class Reference

```
#include <ql/Instruments/stock.hpp>
```

Inheritance diagram for Stock:



7.470.1 Detailed Description

Simple stock class.

Public Member Functions

- **Stock** (const [Handle](#)< [Quote](#) > "e)
- bool [isExpired](#) () const
returns whether the instrument is still tradable.

Protected Member Functions

- void [performCalculations](#) () const

7.470.2 Member Function Documentation

7.470.2.1 void performCalculations () const [protected, virtual]

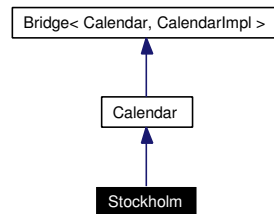
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

7.471 Stockholm Class Reference

```
#include <ql/Calendars/stockholm.hpp>
```

Inheritance diagram for Stockholm:



7.471.1 Detailed Description

Stockholm calendar

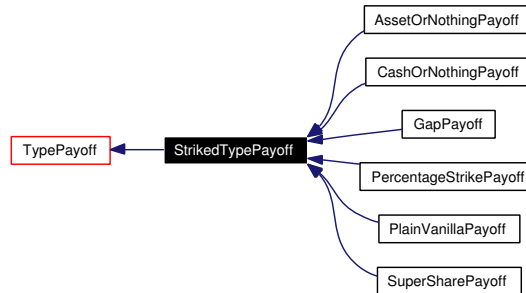
Holidays:

- Saturdays
- Sundays
- Good Friday
- Easter Monday
- Ascension
- Whit(Pentecost) Monday
- Midsummer Eve (Friday between June 18-24)
- New Year's Day, January 1st
- Epiphany, January 6th
- May Day, May 1st
- National Day, June 6th
- Christmas Eve, December 24th
- Christmas Day, December 25th
- Boxing Day, December 26th
- New Year's Eve, December 31th

7.472 StrikedTypePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for StrikedTypePayoff:



7.472.1 Detailed Description

Intermediate class for payoffs based on a fixed strike.

Public Member Functions

- `StrikedTypePayoff` (Option::Type type, Real strike)
- `Real strike` () const

Protected Attributes

- `Real strike_`

7.473 StringFormatter Class Reference

```
#include <ql/basicdataformatters.hpp>
```

7.473.1 Detailed Description

Formats strings as lower- or uppercase.

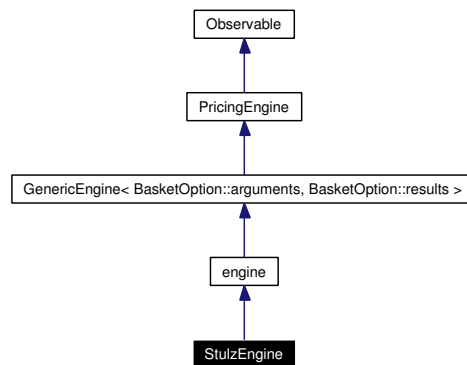
Static Public Member Functions

- `std::string toLowercase (const std::string &s)`
- `std::string toUppercase (const std::string &s)`

7.474 StulzEngine Class Reference

```
#include <ql/PricingEngines/Basket/stulzengine.hpp>
```

Inheritance diagram for StulzEngine:



7.474.1 Detailed Description

Pricing engine for 2D European Baskets.

This class implements formulae from "Options on the Minimum or the Maximum of Two Risky Assets", Rene Stulz, Journal of Financial Economics (1982) 10, 161-185.

Tests

the correctness of the returned value is tested by reproducing results available in literature.

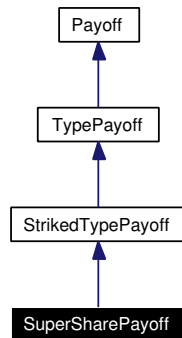
Public Member Functions

- void **calculate** () const

7.475 SuperSharePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for SuperSharePayoff:



7.475.1 Detailed Description

Binary supershare payoff.

Public Member Functions

- **SuperSharePayoff** (Option::Type type, [Real](#) strike, [Real](#) strikeIncrement)
- [Real](#) **operator()** ([Real](#) price) const
- [Real](#) **strikeIncrement** () const

7.476 SVD Class Reference

```
#include <ql/Math/svd.hpp>
```

7.476.1 Detailed Description

Singular value decomposition.

Refer to Golub and Van Loan: [Matrix](#) computation, The Johns Hopkins University Press

Tests

the correctness of the returned values is tested by checking their properties.

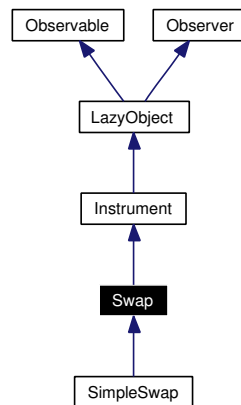
Public Member Functions

- **SVD** (const [Matrix](#) &)
- const [Matrix](#) & **U** () const
- const [Matrix](#) & **V** () const
- const [Array](#) & **singularValues** () const
- [Disposable](#)< [Matrix](#) > **S** () const
- [Real](#) **norm2** ()
- [Real](#) **cond** ()
- [Integer](#) **rank** ()

7.477 Swap Class Reference

```
#include <ql/Instruments/swap.hpp>
```

Inheritance diagram for Swap:



7.477.1 Detailed Description

Interest rate swap.

The cash flows belonging to the first leg are paid; the ones belonging to the second leg are received.

Public Member Functions

- **Swap** (const std::vector< boost::shared_ptr< [CashFlow](#) > > &firstLeg, const std::vector< boost::shared_ptr< [CashFlow](#) > > &secondLeg, const [Handle](#)< [YieldTermStructure](#) > &termStructure)

Instrument interface

- bool [isExpired](#) () const
returns whether the instrument is still tradable.

Additional interface

- [Date](#) [startDate](#) () const
- [Date](#) [maturity](#) () const
- [Real](#) [firstLegBPS](#) () const
- [Real](#) [secondLegBPS](#) () const
- [TimeBasket](#) [sensitivity](#) ([Integer](#) basis=2) const

Protected Member Functions

- void [setupExpired](#) () const
- void [performCalculations](#) () const

Protected Attributes

- `std::vector< boost::shared_ptr< CashFlow > > firstLeg_`
- `std::vector< boost::shared_ptr< CashFlow > > secondLeg_`
- `Handle< YieldTermStructure > termStructure_`
- `Real firstLegBPS_`
- `Real secondLegBPS_`

7.477.2 Member Function Documentation

7.477.2.1 [TimeBasket](#) sensitivity ([Integer](#) *basis* = 2) const

[Bug](#)

this method must still be checked. It is not guaranteed to yield the right results.

7.477.2.2 `void setupExpired () const` [protected, virtual]

This method must leave the instrument in a consistent state when the expiration condition is met.

Reimplemented from [Instrument](#).

7.477.2.3 `void performCalculations () const` [protected, virtual]

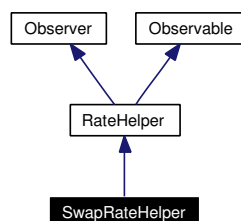
In case a pricing engine is **not** used, this method must be overridden to perform the actual calculations and set any needed results. In case a pricing engine is used, the default implementation can be used.

Reimplemented from [Instrument](#).

7.478 SwapRateHelper Class Reference

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Inheritance diagram for SwapRateHelper:



7.478.1 Detailed Description

Swap rate

Todo

Currency and dayCounter of [Xibor](#) should be added to obtain well define [SwapRateHelper](#)

Warning:

This class assumes that the settlement date does not change between calls of [setTermStructure\(\)](#).

Public Member Functions

- **SwapRateHelper** (const [Handle](#)< [Quote](#) > &rate, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, [Frequency](#) floatingFrequency, [BusinessDayConvention](#) floatingConvention)
- **SwapRateHelper** ([Rate](#) rate, [Integer](#) n, [TimeUnit](#) units, [Integer](#) settlementDays, const [Calendar](#) &calendar, [Frequency](#) fixedFrequency, [BusinessDayConvention](#) fixedConvention, const [DayCounter](#) &fixedDayCount, [Frequency](#) floatingFrequency, [BusinessDayConvention](#) floatingConvention)
- **Real impliedQuote** () const
- **Date latestDate** () const
latest relevant date
- void **setTermStructure** ([YieldTermStructure](#) *)
sets the term structure to be used for pricing

Protected Attributes

- [Integer](#) n_
- [TimeUnit](#) units_
- [Integer](#) settlementDays_
- [Calendar](#) calendar_

- [BusinessDayConvention](#) `fixedConvention_`
- [BusinessDayConvention](#) `floatingConvention_`
- [Frequency](#) `fixedFrequency_`
- [Frequency](#) `floatingFrequency_`
- [DayCounter](#) `fixedDayCount_`
- [Date](#) `settlement_`
- [Date](#) `latestDate_`
- `boost::shared_ptr< SimpleSwap > swap_`
- `Handle< YieldTermStructure > termStructureHandle_`

7.478.2 Member Function Documentation

7.478.2.1 [Date](#) `latestDate () const` [virtual]

latest relevant date

The latest date at which discounts are needed by the helper in order to provide a quote. It does not necessarily equal the maturity of the underlying instrument.

Implements [RateHelper](#).

7.478.2.2 `void setTermStructure (YieldTermStructure *)` [virtual]

sets the term structure to be used for pricing

Warning:

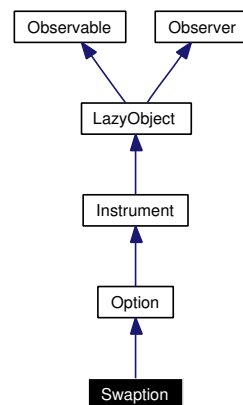
Being a pointer and not a `shared_ptr`, the term structure is not guaranteed to remain allocated for the whole life of the rate helper. It is responsibility of the programmer to ensure that the pointer remains valid. It is advised that rate helpers be used only in term structure constructors, setting the term structure to **this**, i.e., the one being constructed.

Reimplemented from [RateHelper](#).

7.479 Swaption Class Reference

```
#include <ql/Instruments/swaption.hpp>
```

Inheritance diagram for Swaption:



7.479.1 Detailed Description

Swaption class

Tests

- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption decreases (resp. increases) with the strike.
- the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption increases (resp. decreases) with the spread.
- the correctness of the returned value is tested by checking it against that of a swaption on a swap with no spread and a correspondingly adjusted fixed rate.
- the correctness of the returned value is tested by checking it against a known good value.

Public Member Functions

- **Swaption** (const boost::shared_ptr< [SimpleSwap](#) > &swap, const boost::shared_ptr< [Exercise](#) > &exercise, const [Handle](#)< [YieldTermStructure](#) > &termStructure, const boost::shared_ptr< [PricingEngine](#) > &engine)
- bool [isExpired](#) () const
returns whether the instrument is still tradable.
- void [setupArguments](#) ([Arguments](#) *) const

7.479.2 Member Function Documentation

7.479.2.1 void setupArguments ([Arguments](#) *) const [virtual]

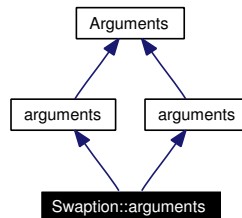
When a derived argument structure is defined for an instrument, this method should be overridden to fill it. This is mandatory in case a pricing engine is used.

Reimplemented from [Instrument](#).

7.480 Swaption::arguments Class Reference

```
#include <ql/Instruments/swaption.hpp>
```

Inheritance diagram for Swaption::arguments:



7.480.1 Detailed Description

Arguments for swaption calculation

Public Member Functions

- `void validate () const`

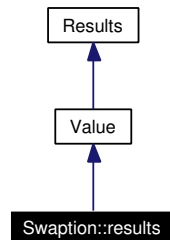
Public Attributes

- [Rate](#) `fairRate`
- [Rate](#) `fixedRate`
- [Real](#) `fixedBPS`

7.481 Swaption::results Class Reference

```
#include <ql/Instruments/swaption.hpp>
```

Inheritance diagram for Swaption::results:



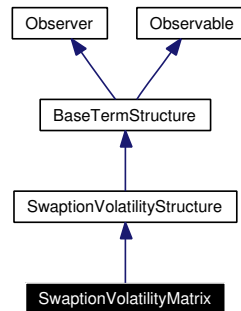
7.481.1 Detailed Description

Results from swaption calculation

7.482 SwaptionVolatilityMatrix Class Reference

```
#include <ql/Volatilities/swaptionvolmatrix.hpp>
```

Inheritance diagram for SwaptionVolatilityMatrix:



7.482.1 Detailed Description

At-the-money swaption-volatility matrix.

This class provides the at-the-money volatility for a given swaption by interpolating a volatility matrix whose elements are the market volatilities of a set of swaption with given exercise date and length.

Todo

either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the exercise date and length vector but an interpolation pointing to the original ones.

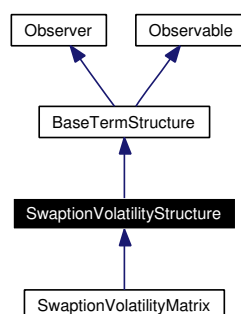
Public Member Functions

- **SwaptionVolatilityMatrix** (const [Date](#) &referenceDate, const std::vector< [Date](#) > &exerciseDates, const std::vector< [Period](#) > &lengths, const [Matrix](#) &volatilities, const [DayCounter](#) &dayCounter)
- [DayCounter](#) dayCounter () const
the day counter used for date/time conversion
- const std::vector< [Date](#) > & exerciseDates () const
- const std::vector< [Period](#) > & lengths () const

7.483 SwaptionVolatilityStructure Class Reference

```
#include <ql/swaptionvolstructure.hpp>
```

Inheritance diagram for SwaptionVolatilityStructure:



7.483.1 Detailed Description

Swaption-volatility structure

This class is purely abstract and defines the interface of concrete swaption volatility structures which will be derived from this one.

Public Member Functions

Constructors

See the BaseTermStructure documentation for issues regarding constructors.

- [SwaptionVolatilityStructure](#) ()
default constructor
- [SwaptionVolatilityStructure](#) (const [Date](#) &today, const [Date](#) &referenceDate)
initialize with a fixed reference date
- [SwaptionVolatilityStructure](#) (const [Date](#) &referenceDate)
initialize with a fixed reference date
- [SwaptionVolatilityStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)
calculate the reference date based on the global evaluation date

Volatility

- [Volatility volatility](#) (const [Date](#) &start, const [Period](#) &length, [Rate](#) strike) const
returns the volatility for a given starting date and length
- [Volatility volatility](#) ([Time](#) start, [Time](#) length, [Rate](#) strike) const
returns the volatility for a given starting time and length

Protected Member Functions

- virtual [Volatility](#) volatilityImpl ([Time](#) start, [Time](#) length, [Rate](#) strike) const =0
implements the actual volatility calculation in derived classes
- virtual std::pair< [Time](#), [Time](#) > convertDates (const [Date](#) &start, const [Period](#) &length) const
implements the conversion between dates and times

7.483.2 Constructor & Destructor Documentation

7.483.2.1 [SwaptionVolatilityStructure](#) ()

default constructor

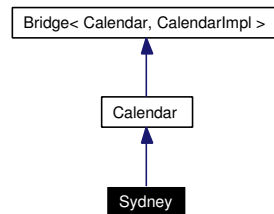
Warning:

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

7.484 Sydney Class Reference

```
#include <ql/Calendars/sydney.hpp>
```

Inheritance diagram for Sydney:



7.484.1 Detailed Description

Sydney calendar (New South Wales, Australia)

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Australia Day, January 26th (possibly moved to Monday)
- Good Friday
- Easter Monday
- ANZAC Day. April 25th (possibly moved to Monday)
- Queen's Birthday, second Monday in June
- Bank Holiday, first Monday in August
- Labour Day, first Monday in October
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

7.485 SymmetricSchurDecomposition Class Reference

```
#include <ql/Math/symmetricschurdecomposition.hpp>
```

7.485.1 Detailed Description

symmetric threshold Jacobi algorithm.

Given a real symmetric matrix S , the Schur decomposition finds the eigenvalues and eigenvectors of S . If D is the diagonal matrix formed by the eigenvalues and U the unitarian matrix of the eigenvectors we can write the Schur decomposition as

$$S = U \cdot D \cdot U^T,$$

where \cdot is the standard matrix product and T is the transpose operator. This class implements the Schur decomposition using the symmetric threshold Jacobi algorithm. For details on the different Jacobi transformations see "Matrix computation," second edition, by Golub and Van Loan, The Johns Hopkins University Press

Tests

the correctness of the returned values is tested by checking their properties.

Public Member Functions

- [SymmetricSchurDecomposition](#) (const [Matrix](#) &s)
- const [Array](#) & **eigenvalues** () const
- const [Matrix](#) & **eigenvectors** () const

7.485.2 Constructor & Destructor Documentation

7.485.2.1 [SymmetricSchurDecomposition](#) (const [Matrix](#) & s)

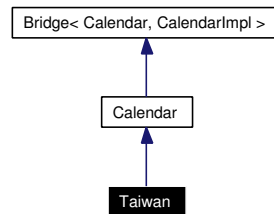
Precondition:

s must be symmetric

7.486 Taiwan Class Reference

```
#include <ql/Calendars/taiwan.hpp>
```

Inheritance diagram for Taiwan:



7.486.1 Detailed Description

Taiwan calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Peace Day, February 28th
- Labor Day, May 1st
- National Day, October 10th

Other holidays for which no rule is given (data available for 2004-2006 only:)

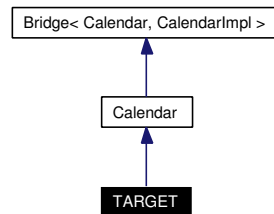
- Lunar New Year
- Tomb Sweeping Day
- Dragon Boat Day
- Mid-Autumn Festival

Data from <http://www.taifex.com.tw>

7.487 TARGET Class Reference

```
#include <ql/Calendars/target.hpp>
```

Inheritance diagram for TARGET:



7.487.1 Detailed Description

TARGET calendar

Holidays (see <http://www.ecb.int>):

- Saturdays
- Sundays
- New Year's Day, January 1st
- Good Friday (since 2000)
- Easter Monday (since 2000)
- Labour Day, May 1st (since 2000)
- Christmas, December 25th
- Day of Goodwill, December 26th (since 2000)
- December 31st (1998, 1999, and 2001)

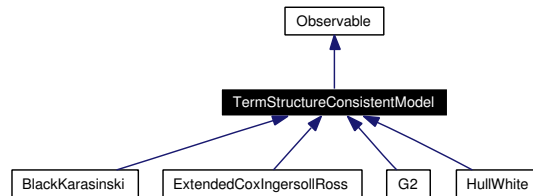
Tests

the correctness of the returned results is tested against a list of known holidays.

7.488 TermStructureConsistentModel Class Reference

```
#include <ql/ShortRateModels/model.hpp>
```

Inheritance diagram for TermStructureConsistentModel:



7.488.1 Detailed Description

Term-structure consistent model class.

This is a base class for models that can reprice exactly any discount bond.

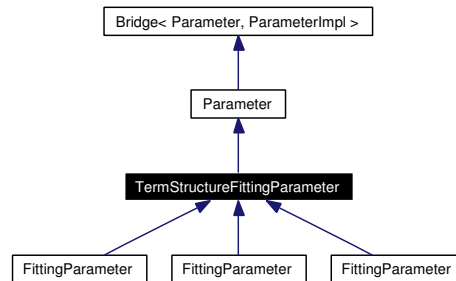
Public Member Functions

- **TermStructureConsistentModel** (const [Handle](#)< [YieldTermStructure](#) > &termStructure)
- const [Handle](#)< [YieldTermStructure](#) > & **termStructure** () const

7.489 TermStructureFittingParameter Class Reference

#include <ql/ShortRateModels/parameter.hpp>

Inheritance diagram for TermStructureFittingParameter:



7.489.1 Detailed Description

Deterministic time-dependent parameter used for yield-curve fitting.

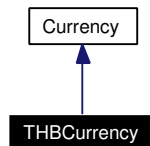
Public Member Functions

- `TermStructureFittingParameter` (const boost::shared_ptr< Parameter::Impl > &impl)
- `TermStructureFittingParameter` (const [Handle](#)< [YieldTermStructure](#) > &term)

7.490 THBCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for THBCurrency:



7.490.1 Detailed Description

Thai baht.

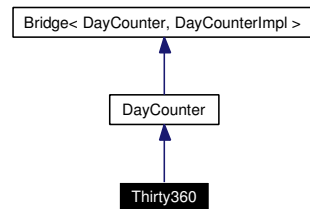
The ISO three-letter code is THB; the numeric code is 764. It is divided in 100 stang.

ingroup currencies

7.491 Thirty360 Class Reference

```
#include <ql/DayCounters/thirty360.hpp>
```

Inheritance diagram for Thirty360:



7.491.1 Detailed Description

30/360 day count convention

The 30/360 day count can be calculated according to US, European, or Italian conventions.

US (NASD) convention: if the starting date is the 31st of a month, it becomes equal to the 30th of the same month. If the ending date is the 31st of a month and the starting date is earlier than the 30th of a month, the ending date becomes equal to the 1st of the next month, otherwise the ending date becomes equal to the 30th of the same month. Also known as "30/360", "360/360", or "Bond Basis"

European convention: starting dates or ending dates that occur on the 31st of a month become equal to the 30th of the same month. Also known as "30E/360", or "Eurobond Basis"

Italian convention: starting dates or ending dates that occur on February and are greater than 27 become equal to 30 for computational sake.

Public Types

- enum **Convention** {
 USA, **BondBasis**, **European**, **EurobondBasis**,
 Italian }

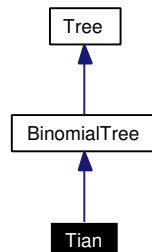
Public Member Functions

- **Thirty360** (Convention c=Thirty360::USA)

7.492 Tian Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for Tian:



7.492.1 Detailed Description

Tian tree: third moment matching, multiplicative approach

Public Member Functions

- **Tian** (const boost::shared_ptr< [StochasticProcess](#) > &process, [Time](#) end, [Size](#) steps, [Real](#) strike)
- [Real](#) underlying ([Size](#) i, [Size](#) index) const
- [Real](#) probability ([Size](#), [Size](#), [Size](#)) const

Protected Attributes

- [Real](#) up_
- [Real](#) down_
- [Real](#) pu_
- [Real](#) pd_

7.493 TimeBasket Class Reference

```
#include <ql/CashFlows/timebasket.hpp>
```

7.493.1 Detailed Description

Distribution over a number of dates.

Map interface

- typedef super::iterator **iterator**
- typedef super::const_iterator **const_iterator**
- typedef super::reverse_iterator **reverse_iterator**
- typedef super::const_reverse_iterator **const_reverse_iterator**
- bool **hasDate** (const [Date](#) &) const

membership

Public Member Functions

- **TimeBasket** (const std::vector< [Date](#) > &dates, const std::vector< [Real](#) > &values)

Algebra

- [TimeBasket](#) & **operator+=** (const [TimeBasket](#) &other)
- [TimeBasket](#) & **operator-=** (const [TimeBasket](#) &other)

Other methods

- [TimeBasket](#) **rebin** (const std::vector< [Date](#) > &buckets) const
- redistribute the entries over the given dates*

7.494 TimeGrid Class Reference

```
#include <ql/grid.hpp>
```

7.494.1 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

7.494.2 Constructor & Destructor Documentation

7.494.2.1 [TimeGrid](#) (Iterator *begin*, Iterator *end*)

Time grid with mandatory time points.

Mandatory points are guaranteed to belong to the grid. No additional points are added.

7.494.2.2 [TimeGrid](#) (Iterator *begin*, Iterator *end*, [Size](#) *steps*)

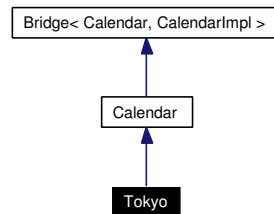
Time grid with mandatory time points.

Mandatory points are guaranteed to belong to the grid. Additional points are then added with regular spacing between pairs of mandatory times in order to reach the desired number of steps.

7.495 Tokyo Class Reference

```
#include <ql/Calendars/tokyo.hpp>
```

Inheritance diagram for Tokyo:



7.495.1 Detailed Description

Tokyo calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Bank Holiday, January 2nd
- Bank Holiday, January 3rd
- Coming of Age Day, 2nd Monday in January
- National Foundation Day, February 11th
- Vernal Equinox
- Greenery Day, April 29th
- Constitution Memorial Day, May 3rd
- Holiday for a Nation, May 4th
- Children's Day, May 5th
- Marine Day, 3rd Monday in July
- Respect for the Aged Day, 3rd Monday in September
- Autumnal Equinox
- Health and Sports Day, 2nd Monday in October
- National Culture Day, November 3rd
- Labor Thanksgiving Day, November 23rd
- Emperor's Birthday, December 23rd
- Bank Holiday, December 31st

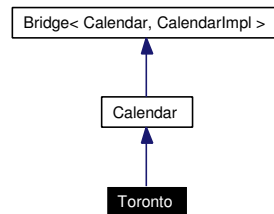
- a few one-shot holidays

Holidays falling on a Sunday are observed on the Monday following except for the bank holidays associated with the new year.

7.496 Toronto Class Reference

```
#include <ql/Calendars/toronto.hpp>
```

Inheritance diagram for Toronto:



7.496.1 Detailed Description

Toronto calendar

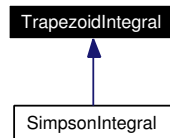
Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Victoria Day, The Monday on or preceding 24 May
- Canada Day, July 1st (possibly moved to Monday)
- Provincial Holiday, first Monday of August
- Labour Day, first Monday of September
- Thanksgiving Day, second Monday of October
- Remembrance Day, November 11th
- Christmas, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

7.497 TrapezoidIntegral Class Reference

```
#include <ql/Math/trapezoidintegral.hpp>
```

Inheritance diagram for TrapezoidIntegral:



7.497.1 Detailed Description

Integral of a one-dimensional function.

Given a target accuracy ϵ , the integral of a function f between a and b is calculated by means of the trapezoid formula

$$\int_a^b f dx = \frac{1}{2}f(x_0) + f(x_1) + f(x_2) + \dots + f(x_{N-1}) + \frac{1}{2}f(x_N)$$

where $x_0 = a$, $x_N = b$, and $x_i = a + i\Delta x$ with $\Delta x = (b - a)/N$. The number N of intervals is repeatedly increased until the target accuracy is reached.

Tests

the correctness of the result is tested by checking it against known good values.

Public Types

- enum **Method** { **Default**, **MidPoint** }

Public Member Functions

- **TrapezoidIntegral** ([Real](#) accuracy, Method method=Default, [Size](#) maxIterations=[Null](#)< [Size](#) >())
- template<class F> [Real](#) **operator()** (const F &f, [Real](#) a, [Real](#) b) const
- [Real](#) **accuracy** () const
- [Real](#) & **accuracy** ()
- Method **method** () const
- Method & **method** ()
- [Size](#) **maxIterations** () const
- [Size](#) & **maxIterations** ()

Protected Member Functions

- template<class F> [Real](#) **defaultIteration** (const F &f, [Real](#) a, [Real](#) b, [Real](#) I, [Size](#) N) const
- template<class F> [Real](#) **midPointIteration** (const F &f, [Real](#) a, [Real](#) b, [Real](#) I, [Size](#) N) const

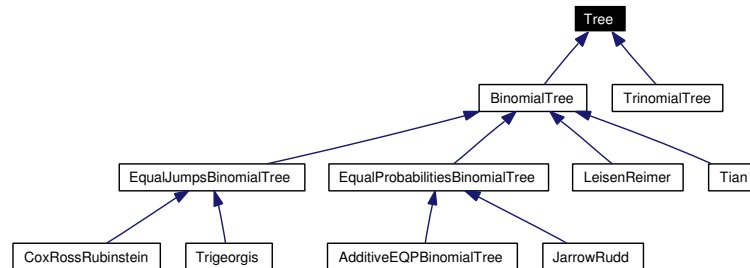
Protected Attributes

- [Real](#) accuracy_
- Method method_
- [Size](#) maxIterations_

7.498 Tree Class Reference

```
#include <ql/Lattices/tree.hpp>
```

Inheritance diagram for Tree:



7.498.1 Detailed Description

Tree approximating a single-factor diffusion

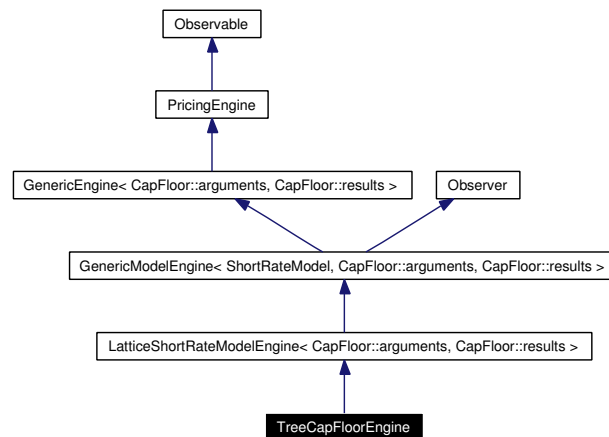
Public Member Functions

- **Tree** ([Size](#) nColumns)
- virtual [Real](#) **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 [Real](#) **probability** ([Size](#) i, [Size](#) index, [Size](#) branch) const =0
- [Size](#) **nColumns** () const

7.499 TreeCapFloorEngine Class Reference

```
#include <ql/PricingEngines/CapFloor/treecapfloorengine.hpp>
```

Inheritance diagram for TreeCapFloorEngine:



7.499.1 Detailed Description

Numerical lattice engine for cap/floors.

Bug

caplets which have already fixed are not included in the cap value.

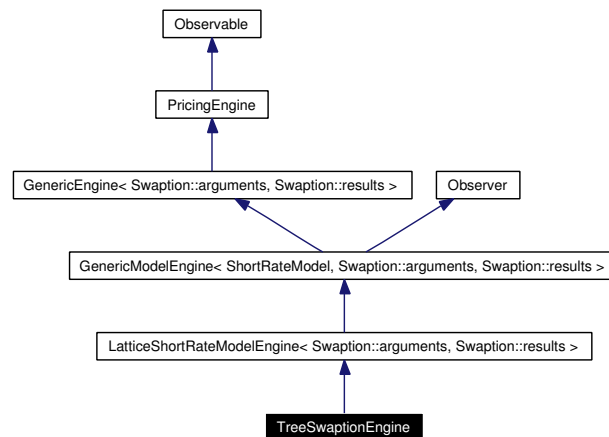
Public Member Functions

- **TreeCapFloorEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, [Size](#) timeSteps)
- **TreeCapFloorEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

7.500 TreeSwaptionEngine Class Reference

```
#include <ql/PricingEngines/Swaption/treeswaptionengine.hpp>
```

Inheritance diagram for TreeSwaptionEngine:



7.500.1 Detailed Description

Numerical lattice engine for swaptions.

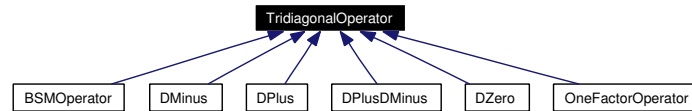
Public Member Functions

- **TreeSwaptionEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, [Size](#) timeSteps)
- **TreeSwaptionEngine** (const boost::shared_ptr< [ShortRateModel](#) > &model, const [TimeGrid](#) &timeGrid)
- void **calculate** () const

7.501 TridiagonalOperator Class Reference

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Inheritance diagram for TridiagonalOperator:



7.501.1 Detailed Description

Base implementation for tridiagonal operator.

Warning:

to use real time-dependant algebra, you must overload the corresponding operators in the inheriting time-dependent class

Operator interface

- `Disposable< Array > applyTo (const Array &v) const`
apply operator to a given array
- `Disposable< Array > solveFor (const Array &rhs) const`
solve linear system for a given right-hand side
- `Disposable< Array > SOR (const Array &rhs, Real tol) const`
solve linear system with SOR approach
- `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** ([Real](#), [Real](#))
- void **setMidRow** ([Size](#), [Real](#), [Real](#), [Real](#))
- void **setMidRows** ([Real](#), [Real](#), [Real](#))
- void **setLastRow** ([Real](#), [Real](#))
- void **setTime** ([Time](#) t)

Utilities

- void **swap** ([TridiagonalOperator](#) &)

Protected Attributes

- [Array](#) **diagonal_**
- [Array](#) **lowerDiagonal_**
- [Array](#) **upperDiagonal_**
- boost::shared_ptr< [TimeSetter](#) > **timeSetter_**

Friends

- [Disposable](#)< [TridiagonalOperator](#) > **operator+** (const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator-** (const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator+** (const [TridiagonalOperator](#) &, const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator-** (const [TridiagonalOperator](#) &, const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator** * ([Real](#), const [TridiagonalOperator](#) &)
- [Disposable](#)< [TridiagonalOperator](#) > **operator** * (const [TridiagonalOperator](#) &, [Real](#))
- [Disposable](#)< [TridiagonalOperator](#) > **operator** / (const [TridiagonalOperator](#) &, [Real](#))

7.502 TridiagonalOperator::TimeSetter Class Reference

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

7.502.1 Detailed Description

encapsulation of time-setting logic

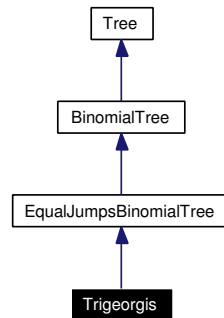
Public Member Functions

- virtual void **setTime** ([Time](#) t, [TridiagonalOperator](#) &L) const =0

7.503 Trigeorgis Class Reference

```
#include <ql/Lattices/binomialtree.hpp>
```

Inheritance diagram for Trigeorgis:



7.503.1 Detailed Description

Trigeorgis (additive equal jumps) binomial tree

Public Member Functions

- **Trigeorgis** (const boost::shared_ptr< [StochasticProcess](#) > &process, [Time](#) end, [Size](#) steps, [Real](#) strike)

7.504 TrinomialBranching Class Reference

```
#include <ql/Lattices/trinomialtree.hpp>
```

7.504.1 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
- [Real](#) probability ([Size](#) index, [Size](#) branch) const
- [Integer](#) jMin () const

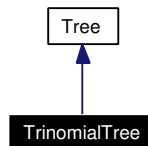
Friends

- class [TrinomialTree](#)

7.505 TrinomialTree Class Reference

```
#include <ql/Lattices/trinomialtree.hpp>
```

Inheritance diagram for TrinomialTree:



7.505.1 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 boost::shared_ptr< [StochasticProcess](#) > &process, const [TimeGrid](#) &timeGrid, bool isPositive=false)
- [Real](#) **dx** ([Size](#) i) const
- [Size](#) **size** ([Size](#) i) const
- [Real](#) **underlying** ([Size](#) i, [Size](#) index) const
- const [TimeGrid](#) & **timeGrid** () const
- [Size](#) **descendant** ([Size](#) i, [Size](#) index, [Size](#) branch) const
- [Real](#) **probability** ([Size](#) i, [Size](#) index, [Size](#) branch) const

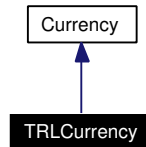
Protected Attributes

- std::vector< boost::shared_ptr< [TrinomialBranching](#) > > **branchings_**
- [Real](#) **x0_**
- std::vector< [Real](#) > **dx_**
- [TimeGrid](#) **timeGrid_**

7.506 TRLCurrency Class Reference

```
#include <ql/Currencies/europe.hpp>
```

Inheritance diagram for TRLCurrency:



7.506.1 Detailed Description

Turkish lira.

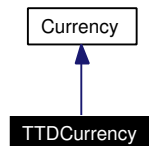
The ISO three-letter code is TRL; the numeric code is 792. It is divided in 100 kurus.

ingroup currencies

7.507 TTDCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for TTDCurrency:



7.507.1 Detailed Description

Trinidad & Tobago dollar.

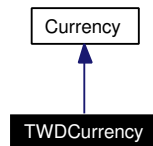
The ISO three-letter code is TTD; the numeric code is 780. It is divided in 100 cents.

ingroup currencies

7.508 TWDCurrency Class Reference

```
#include <ql/Currencies/asia.hpp>
```

Inheritance diagram for TWDCurrency:



7.508.1 Detailed Description

[Taiwan](#) dollar.

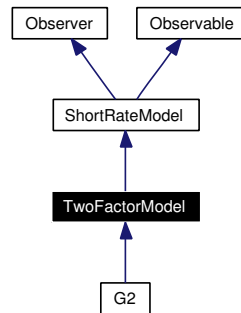
The ISO three-letter code is TWD; the numeric code is 901. It is divided in 100 cents.

ingroup currencies

7.509 TwoFactorModel Class Reference

```
#include <ql/ShortRateModels/twofactormodel.hpp>
```

Inheritance diagram for TwoFactorModel:



7.509.1 Detailed Description

Abstract base-class for two-factor models.

Public Member Functions

- **TwoFactorModel** ([Size](#) nParams)
- virtual `boost::shared_ptr< ShortRateDynamics > dynamics ()` const =0
Returns the short-rate dynamics.
- virtual `boost::shared_ptr< Lattice > tree (const TimeGrid &grid)` const
Returns a two-dimensional trinomial tree.

7.510 TwoFactorModel::ShortRateDynamics Class Reference

```
#include <ql/ShortRateModels/twofactormodel.hpp>
```

7.510.1 Detailed Description

Class describing the dynamics of the two state variables.

We assume here that the short-rate is a function of two state variables x and y .

$$r_t = f(t, x_t, y_t)$$

of two state variables x_t and y_t . These stochastic processes satisfy

$$x_t = \mu_x(t, x_t)dt + \sigma_x(t, x_t)dW_t^x$$

and

$$y_t = \mu_y(t, y_t)dt + \sigma_y(t, y_t)dW_t^y$$

where W^x and W^y are two brownian motions satisfying

$$dW_t^x dW_t^y = \rho dt$$

.

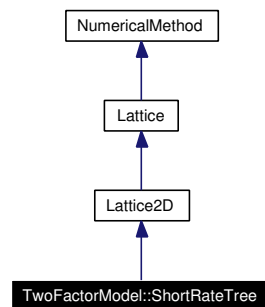
Public Member Functions

- **ShortRateDynamics** (const boost::shared_ptr< [StochasticProcess](#) > &xProcess, const boost::shared_ptr< [StochasticProcess](#) > &yProcess, [Real](#) correlation)
- virtual [Rate](#) **shortRate** ([Time](#) t, [Real](#) x, [Real](#) y) const =0
- const boost::shared_ptr< [StochasticProcess](#) > & **xProcess** () const
Risk-neutral dynamics of the first state variable x.
- const boost::shared_ptr< [StochasticProcess](#) > & **yProcess** () const
Risk-neutral dynamics of the second state variable y.
- [Real](#) **correlation** () const
Correlation ρ between the two brownian motions.

7.511 TwoFactorModel::ShortRateTree Class Reference

```
#include <ql/ShortRateModels/twofactormodel.hpp>
```

Inheritance diagram for TwoFactorModel::ShortRateTree:



7.511.1 Detailed Description

Recombining two-dimensional tree discretizing the state variable.

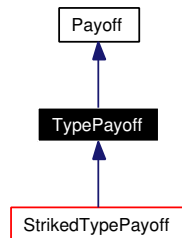
Public Member Functions

- [ShortRateTree](#) (const boost::shared_ptr< [TrinomialTree](#) > &tree1, const boost::shared_ptr< [TrinomialTree](#) > &tree2, const boost::shared_ptr< [ShortRateDynamics](#) > &dynamics)
Plain tree build-up from short-rate dynamics.
- [DiscountFactor](#) [discount](#) ([Size](#) i, [Size](#) index) const
Discount factor at time t_i and node indexed by index.

7.512 TypePayoff Class Reference

```
#include <ql/Instruments/payoffs.hpp>
```

Inheritance diagram for TypePayoff:



7.512.1 Detailed Description

Intermediate class for call/put/straddle payoffs.

Public Member Functions

- **TypePayoff** (Option::Type type)
- Option::Type **optionType** () const

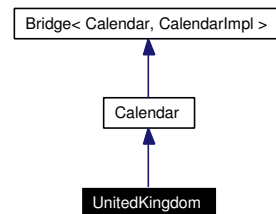
Protected Attributes

- Option::Type **type_**

7.513 UnitedKingdom Class Reference

```
#include <ql/Calendars/unitedkingdom.hpp>
```

Inheritance diagram for UnitedKingdom:



7.513.1 Detailed Description

United Kingdom calendars.

Public holidays (data from <http://www.dti.gov.uk/er/bankhol.htm>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Early May Bank Holiday, first Monday of May
- Spring Bank Holiday, last Monday of May
- Summer Bank Holiday, last Monday of August
- Christmas Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Holidays for the stock exchange:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Early May Bank Holiday, first Monday of May
- Spring Bank Holiday, last Monday of May
- Summer Bank Holiday, last Monday of August

- Christmas Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Holidays for the metals exchange:

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday)
- Good Friday
- Easter Monday
- Early May Bank Holiday, first Monday of May
- Spring Bank Holiday, last Monday of May
- Summer Bank Holiday, last Monday of August
- Christmas Day, December 25th (possibly moved to Monday or Tuesday)
- Boxing Day, December 26th (possibly moved to Monday or Tuesday)

Todo

add LIFFE

Tests

the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum [Market](#) { [Settlement](#), [Exchange](#), [Metals](#) }
UK calendars.

Public Member Functions

- [UnitedKingdom](#) ([Market](#) market=Settlement)

7.513.2 Member Enumeration Documentation

7.513.2.1 enum [Market](#)

UK calendars.

Enumeration values:

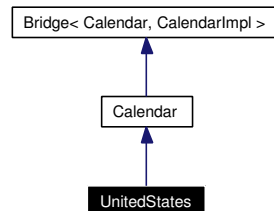
Settlement generic settlement calendar

Exchange London stock-exchange calendar.

7.514 UnitedStates Class Reference

```
#include <ql/Calendars/unitedstates.hpp>
```

Inheritance diagram for UnitedStates:



7.514.1 Detailed Description

United States calendars.

Public holidays (see: <http://www.opm.gov/fedhol/>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday, or to Friday if on Saturday)
- Martin Luther King's birthday, third Monday in January
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Columbus Day, second Monday in October
- Veterans' Day, November 11th (moved to Monday if Sunday or Friday if Saturday)
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

Holidays for the stock exchange (data from <http://www.nyse.com>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Martin Luther King's birthday, third Monday in January
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Good Friday

- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

Holidays for the government bond market (data from <http://www.bondmarkets.com>):

- Saturdays
- Sundays
- New Year's Day, January 1st (possibly moved to Monday if actually on Sunday)
- Martin Luther King's birthday, third Monday in January
- Presidents' Day (a.k.a. Washington's birthday), third Monday in February
- Good Friday
- Memorial Day, last Monday in May
- Independence Day, July 4th (moved to Monday if Sunday or Friday if Saturday)
- Labor Day, first Monday in September
- Columbus Day, second Monday in October
- Veterans' Day, November 11th (moved to Monday if Sunday or Friday if Saturday)
- Thanksgiving Day, fourth Thursday in November
- Christmas, December 25th (moved to Monday if Sunday or Friday if Saturday)

Tests

the correctness of the returned results is tested against a list of known holidays.

Public Types

- enum `Market` { `Settlement`, `Exchange`, `GovernmentBond` }
US calendars.

Public Member Functions

- `UnitedStates` (`Market` market=`Settlement`)

7.514.2 Member Enumeration Documentation

7.514.2.1 enum [Market](#)

US calendars.

Enumeration values:

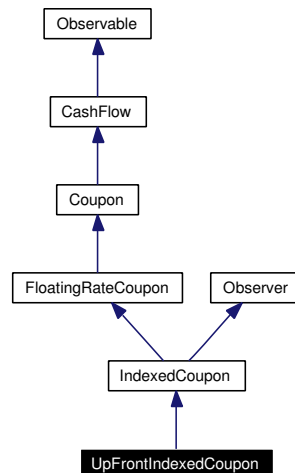
Settlement generic settlement calendar

Exchange New York stock-exchange calendar.

7.515 UpFrontIndexedCoupon Class Reference

```
#include <ql/CashFlows/upfrontindexedcoupon.hpp>
```

Inheritance diagram for UpFrontIndexedCoupon:



7.515.1 Detailed Description

up front indexed coupon class

Warning:

This class does not perform any date adjustment, i.e., the start and end date passed upon construction should be already rolled to a business day.

Public Member Functions

- **UpFrontIndexedCoupon** (*Real* nominal, const *Date* &paymentDate, const boost::shared_ptr< *Xibor* > &index, const *Date* &startDate, const *Date* &endDate, *Integer* fixingDays, *Spread* spread=0.0, const *Date* &refPeriodStart=*Date*(), const *Date* &refPeriodEnd=*Date*(), const *DayCounter* &dayCounter=*DayCounter*())

FloatingRateCoupon interface

- *Date* *fixingDate* () const
fixing date

Visitability

- virtual void **accept** (*AcyclicVisitor* &)

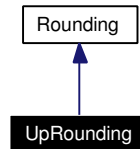
Protected Attributes

- *Calendar* *calendar_*

7.516 UpRounding Class Reference

```
#include <ql/Math/rounding.hpp>
```

Inheritance diagram for UpRounding:



7.516.1 Detailed Description

Up-rounding.

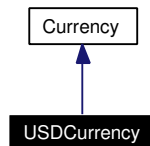
Public Member Functions

- **UpRounding** ([Integer](#) precision, [Integer](#) digit=5)

7.517 USDCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for USDCurrency:



7.517.1 Detailed Description

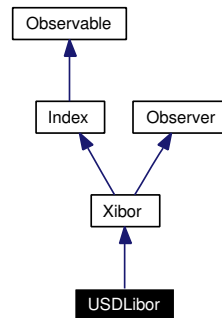
U.S. dollar.

The ISO three-letter code is USD; the numeric code is 840. It is divided in 100 cents.

7.518 USDLibor Class Reference

```
#include <ql/Indexes/usdlibor.hpp>
```

Inheritance diagram for USDLibor:



7.518.1 Detailed Description

USD Libor index

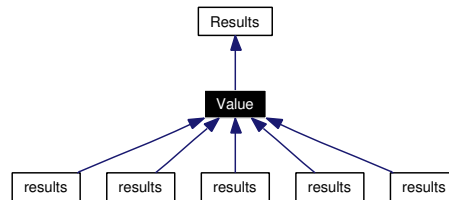
Public Member Functions

- **USDLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual360](#)())

7.519 Value Class Reference

```
#include <ql/instrument.hpp>
```

Inheritance diagram for Value:



7.519.1 Detailed Description

pricing results

Public Member Functions

- `void reset ()`

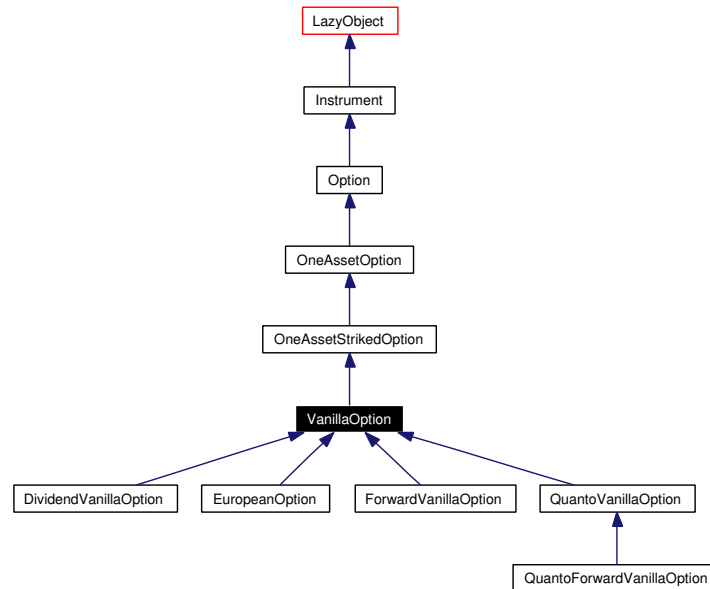
Public Attributes

- [Real](#) value
- [Real](#) errorEstimate

7.520 VanillaOption Class Reference

```
#include <ql/Instruments/vanillaoption.hpp>
```

Inheritance diagram for VanillaOption:



7.520.1 Detailed Description

Vanilla option (no discrete dividends, no barriers) on a single asset.

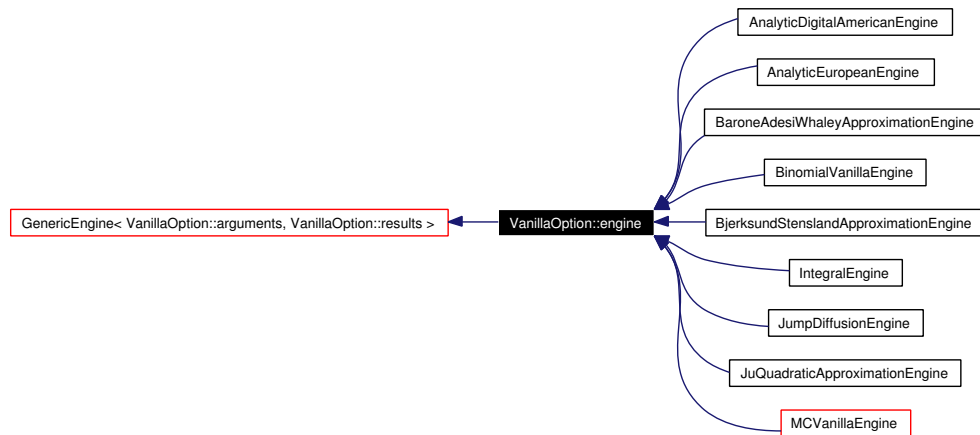
Public Member Functions

- **VanillaOption** (const boost::shared_ptr< [BlackScholesProcess](#) > &, const boost::shared_ptr< [StrikedTypePayoff](#) > &payoff, const boost::shared_ptr< [Exercise](#) > &exercise, const boost::shared_ptr< [PricingEngine](#) > &engine=boost::shared_ptr< [PricingEngine](#) >())

7.521 VanillaOption::engine Class Reference

```
#include <ql/Instruments/vanillaoption.hpp>
```

Inheritance diagram for VanillaOption::engine:



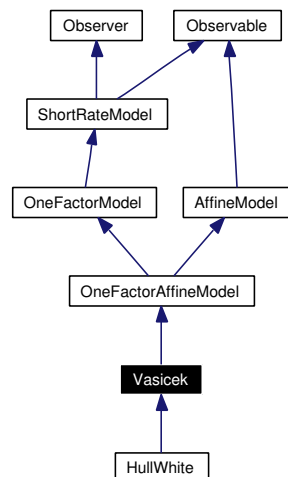
7.521.1 Detailed Description

Vanilla option engine base class.

7.522 Vasicek Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>
```

Inheritance diagram for Vasicek:



7.522.1 Detailed Description

Vasicek model class

This class implements the [Vasicek](#) model defined by

$$dr_t = a(b - r_t)dt + \sigma dW_t,$$

where a , b and σ are constants.

Public Member Functions

- **Vasicek** ([Rate](#) r0=0.05, [Real](#) a=0.1, [Real](#) b=0.05, [Real](#) sigma=0.01)
- virtual [Real](#) **discountBondOption** (Option::Type type, [Real](#) strike, [Time](#) maturity, [Time](#) bondMaturity) const
- virtual boost::shared_ptr< ShortRateDynamics > **dynamics** () const
returns the short-rate dynamics

Protected Member Functions

- virtual [Real](#) **A** ([Time](#) t, [Time](#) T) const
- virtual [Real](#) **B** ([Time](#) t, [Time](#) T) const
- [Real](#) **a** () const
- [Real](#) **b** () const
- [Real](#) **sigma** () const

7.523 Vasicek::Dynamics Class Reference

```
#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>
```

7.523.1 Detailed Description

Short-rate dynamics in the Vasicek model.

The short-rate follows an Ornstein-Uhlenbeck process with mean b .

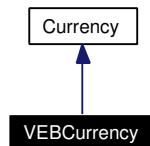
Public Member Functions

- **Dynamics** ([Real](#) a, [Real](#) b, [Real](#) sigma, [Real](#) r0)
- virtual [Real](#) **variable** ([Time](#), [Rate](#) r) const
- virtual [Real](#) **shortRate** ([Time](#), [Real](#) x) const

7.524 VEBCurrency Class Reference

```
#include <ql/Currencies/america.hpp>
```

Inheritance diagram for VEBCurrency:



7.524.1 Detailed Description

Venezuelan bolivar.

The ISO three-letter code is VEB; the numeric code is 862. It is divided in 100 centimos.

ingroup currencies

7.525 Visitor Class Template Reference

```
#include <ql/Patterns/visitor.hpp>
```

7.525.1 Detailed Description

```
template<class T> class QuantLib::Visitor< T >
```

Visitor for a specific class

Public Member Functions

- virtual void **visit** (T &)=0

7.526 VolatilityFormatter Class Reference

```
#include <ql/basicdataformatters.hpp>
```

7.526.1 Detailed Description

Formats volatilities for output.

Formatting is in percentage form (xx.xxxxx)

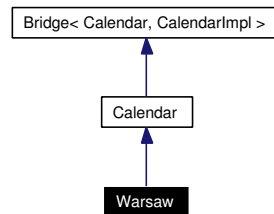
Static Public Member Functions

- `std::string toString` ([Volatility](#) vol, [Integer](#) precision=5)

7.527 Warsaw Class Reference

```
#include <ql/Calendars/warsaw.hpp>
```

Inheritance diagram for Warsaw:



7.527.1 Detailed Description

Warsaw calendar

Holidays:

- Saturdays
- Sundays
- Easter Monday
- Corpus Christi
- New Year's Day, January 1st
- May Day, May 1st
- Constitution Day, May 3rd
- Assumption of the Blessed Virgin Mary, August 15th
- All Saints Day, November 1st
- Independence Day, November 11th
- Christmas, December 25th
- 2nd Day of Christmas, December 26th

7.528 WeekdayFormatter Class Reference

```
#include <ql/date.hpp>
```

7.528.1 Detailed Description

Formats weekday for output.

Formatting can be in Long (full name), [Short](#) (three letters), of Shortest (two letters) form.

Public Types

- enum Format { Long, Short, Shortest }

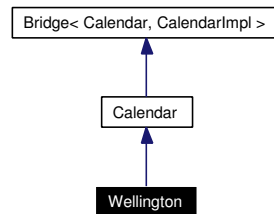
Static Public Member Functions

- std::string toString ([Weekday](#) wd, Format f=Long)

7.529 Wellington Class Reference

```
#include <ql/Calendars/wellington.hpp>
```

Inheritance diagram for Wellington:



7.529.1 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)

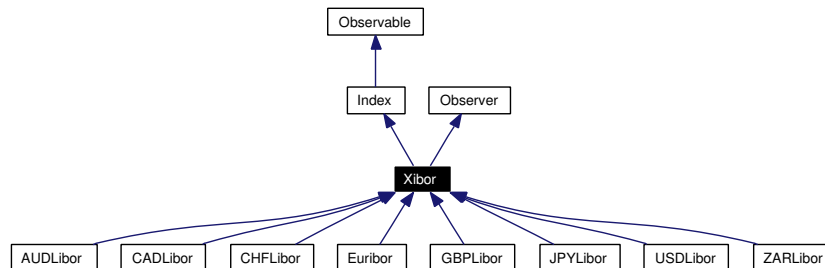
Note:

The holiday rules for [Wellington](#) were documented by David Gilbert for IDB (<http://www.jrefinery.com/ibd/>)

7.530 Xibor Class Reference

```
#include <ql/Indexes/xibor.hpp>
```

Inheritance diagram for Xibor:



7.530.1 Detailed Description

base class for libor indexes

Public Member Functions

- **Xibor** (const std::string &familyName, Integer n, TimeUnit units, Integer settlementDays, const Currency ¤cy, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter, const Handle< YieldTermStructure > &h)
- **Xibor** (const std::string &familyName, Integer n, TimeUnit units, Integer settlementDays, CurrencyTag currency, const Calendar &calendar, BusinessDayConvention convention, const DayCounter &dayCounter, const Handle< YieldTermStructure > &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
- **Frequency frequency** () const
- **Integer settlementDays** () const
- const **Currency & currency** () const
- **Calendar calendar** () const
- bool **isAdjusted** () const
- **BusinessDayConvention businessDayConvention** () const
- **DayCounter dayCounter** () const
- boost::shared_ptr< YieldTermStructure > **termStructure** () const

7.530.2 Constructor & Destructor Documentation

7.530.2.1 **Xibor** (const std::string & *familyName*, **Integer** *n*, **TimeUnit** *units*, **Integer** *settlementDays*, **CurrencyTag** *currency*, const **Calendar** & *calendar*, **BusinessDayConvention** *convention*, const **DayCounter** & *dayCounter*, const **Handle**< **YieldTermStructure** > & *h*)

Deprecated

use the constructor taking a **Currency** instance

7.530.3 Member Function Documentation

7.530.3.1 **Rate** fixing (const **Date** & *fixingDate*) const [virtual]

returns the fixing at the given date

Note:

any date passed as arguments must be a value date, i.e., the real calendar date advanced by a number of settlement days.

Implements **Index**.

7.530.3.2 **void update ()** [virtual]

This method must be implemented in derived classes. An instance of **Observer** does not call this method directly: instead, it will be called by the observables the instance registered with when they need to notify any changes.

Implements **Observer**.

7.530.3.3 **std::string name ()** const [virtual]

Returns the name of the index.

Warning:

This method is used for output and comparison between indexes. It is **not** meant to be used for writing switch-on-type code.

Implements **Index**.

7.531 XiborManager Class Reference

```
#include <ql/Indexes/xibormanager.hpp>
```

7.531.1 Detailed Description

global repository for libor histories

Deprecated

use [IndexManager](#) instead

Public Types

- typedef std::map< std::string, [History](#) > **HistoryMap**

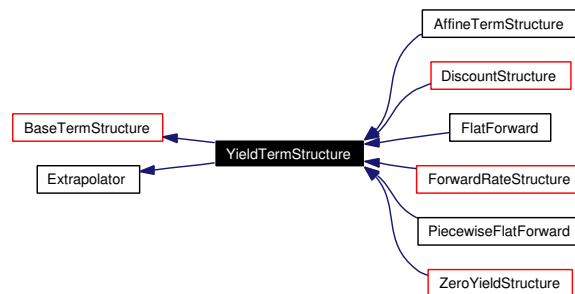
Static Public Member Functions

- 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** ()

7.532 YieldTermStructure Class Reference

```
#include <ql/termstructure.hpp>
```

Inheritance diagram for YieldTermStructure:



7.532.1 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](#), [ForwardRateStructure](#)

Tests

observability against evaluation date changes is checked.

Public Member Functions

- virtual [DayCounter](#) [dayCounter](#) () const =0
the day counter used for date/time conversion

Constructors

See the [BaseTermStructure](#) documentation for issues regarding constructors.

- [YieldTermStructure](#) (const [Date](#) &today'sDate, const [Date](#) &referenceDate)
initialize with a fixed today's date and reference date
- [YieldTermStructure](#) ()
default constructor
- [YieldTermStructure](#) (const [Date](#) &referenceDate)
initialize with a fixed reference date
- [YieldTermStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)

calculate the reference date based on the global evaluation date

zero rates

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](#) t, bool extrapolate=false) const
zero-yield rate
- [Rate zeroCoupon](#) (const [Date](#) &, [Integer](#), bool extrapolate=false) const
zero-coupon rate
- [Rate zeroCoupon](#) ([Time](#), [Integer](#), bool extrapolate=false) const
zero-coupon rate
- [InterestRate zeroRate](#) (const [Date](#) &d, const [DayCounter](#) &resultDayCounter, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const
zero-yield rate
- [InterestRate zeroRate](#) ([Time](#) t, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const
zero-yield rate

discount factors

These methods are either function of dates or times. In the latter case, times are calculated as fraction of year from the reference date.

- [DiscountFactor discount](#) (const [Date](#) &, bool extrapolate=false) const
discount factor
- [DiscountFactor discount](#) ([Time](#), bool extrapolate=false) const
discount factor

forward rates

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 compoundForward](#) (const [Date](#) &d1, [Integer](#) f, bool extrapolate=false) const
instantaneous forward rate at a given compounding frequency
- [Rate compoundForward](#) ([Time](#) t1, [Integer](#) f, bool extrapolate=false) const
instantaneous forward rate at a given compounding frequency
- [Rate instantaneousForward](#) (const [Date](#) &, bool extrapolate=false) const
instantaneous forward rate
- [Rate instantaneousForward](#) ([Time](#), bool extrapolate=false) const
instantaneous forward rate

- **Rate forward** (const [Date](#) &d1, const [Date](#) &d2, bool extrapolate=false) const
discrete forward rate between two dates
- **Rate forward** ([Time](#), [Time](#), bool extrapolate=false) const
discrete forward rate between two times
- **InterestRate forwardRate** (const [Date](#) &d1, const [Date](#) &d2, const [DayCounter](#) &result-DayCounter, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const
forward interest rate
- **InterestRate forwardRate** ([Time](#) t1, [Time](#) t2, Compounding comp, [Frequency](#) freq=Annual, bool extrapolate=false) const
forward interest rate

par rates

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 parRate** ([Year](#) tenor, const [Date](#) &effectiveDate, [Frequency](#) freq=Annual, bool extrapolate=false) const
par rate
- **Rate parRate** ([Year](#) tenor, [Time](#) t0, [Frequency](#) freq=Annual, bool extrapolate=false) const
par rate

Dates

- virtual [Date](#) **maxDate** () const =0
the latest date for which the curve can return rates
- virtual [Time](#) **maxTime** () const
the latest time for which the curve can return rates

Protected Member Functions

Calculations

These methods must be implemented in derived classes to perform the actual discount and rate calculations. When they are called, range check has already been performed; therefore, they must assume that extrapolation is required.

- virtual [DiscountFactor](#) **discountImpl** ([Time](#)) const =0
discount calculation
- virtual [Rate](#) **zeroYieldImpl** ([Time](#)) const =0
zero-yield calculation
- virtual [Rate](#) **forwardImpl** ([Time](#)) const =0
instantaneous forward-rate calculation
- virtual [Rate](#) **compoundForwardImpl** ([Time](#), [Integer](#)) const =0
compound forward-rate calculation

7.532.2 Constructor & Destructor Documentation

7.532.2.1 [YieldTermStructure](#) (const [Date](#) & *today'sDate*, const [Date](#) & *referenceDate*)

initialize with a fixed today's date and reference date

Deprecated

use the constructor without today's date; set the evaluation date through [Settings::instance\(\)](#).

7.532.2.2 [YieldTermStructure](#) ()

default constructor

Warning:

term structures initialized by means of this constructor must manage their own reference date by overriding the [referenceDate\(\)](#) method.

7.532.3 Member Function Documentation

7.532.3.1 [Rate](#) zeroYield (const [Date](#) &, bool *extrapolate* = false) const

zero-yield rate

Deprecated

use zeroRate(const [Date](#)& d, const [DayCounter](#)& dc, Continuous, NoFrequency, bool extrapolate) instead

7.532.3.2 [Rate](#) zeroYield ([Time](#) t, bool *extrapolate* = false) const

zero-yield rate

Deprecated

use zeroRate([Time](#) t, Continuous, NoFrequency, bool extrapolate) instead

7.532.3.3 [Rate](#) zeroCoupon (const [Date](#) &, [Integer](#), bool *extrapolate* = false) const

zero-coupon rate

Deprecated

use zeroRate(const [Date](#)&, const [DayCounter](#)&, Compounding, Frequency, bool) instead

7.532.3.4 [Rate](#) zeroCoupon ([Time](#), [Integer](#), bool *extrapolate* = false) const

zero-coupon rate

Deprecated

use zeroRate([Time](#), Compounding, Frequency, bool) instead

7.532.3.5 [InterestRate](#) `zeroRate (const Date & d, const DayCounter & resultDayCounter, Compounding comp, Frequency freq = Annual, bool extrapolate = false) const`

zero-yield rate

returns the implied zero-yield rate for a given date. The resulting [InterestRate](#) has the required daycounting rule.

7.532.3.6 [InterestRate](#) `zeroRate (Time t, Compounding comp, Frequency freq = Annual, bool extrapolate = false) const`

zero-yield rate

returns the implied zero-yield rate for a given time. The resulting [InterestRate](#) has the same day-counting rule used by the term structure. The same rule should be used for the calculating the time t.

7.532.3.7 [Rate](#) `compoundForward (const Date & d1, Integer f, bool extrapolate = false) const`

instantaneous forward rate at a given compounding frequency

Deprecated

use `forwardRate(const Date& d1, const Date& d1, const DayCounter& dc, SimpleThenCompounded, Frequency f, bool extrapolate)` instead

7.532.3.8 [Rate](#) `compoundForward (Time t1, Integer f, bool extrapolate = false) const`

instantaneous forward rate at a given compounding frequency

Deprecated

use `forwardRate(Time t1, Time t1, SimpleThenCompounded, Frequency f, bool extrapolate)` instead

7.532.3.9 [Rate](#) `instantaneousForward (const Date &, bool extrapolate = false) const`

instantaneous forward rate

Deprecated

use `forwardRate(const Date& d1, const Date& d1, const DayCounter& dc, Continuous, NoFrequency, bool extrapolate)` instead

7.532.3.10 [Rate](#) `instantaneousForward (Time, bool extrapolate = false) const`

instantaneous forward rate

Deprecated

use `forwardRate(Time t1, Time t1, Continuous, NoFrequency, bool extrapolate)` instead

7.532.3.11 Rate forward (const Date & d1, const Date & d2, bool extrapolate = false) const

discrete forward rate between two dates

Deprecated

use forwardRate(const Date& d1, const Date& d2, const DayCounter& dc, Continuous, NoFrequency, bool extrapolate) instead

7.532.3.12 Rate forward (Time, Time, bool extrapolate = false) const

discrete forward rate between two times

Deprecated

use forwardRate(Time t1, Time t2, Continuous, NoFrequency, bool extrapolate) instead

7.532.3.13 InterestRate forwardRate (const Date & d1, const Date & d2, const DayCounter & resultDayCounter, Compounding comp, Frequency freq = Annual, bool extrapolate = false) const

forward interest rate

returns the implied forward interest rate between two dates The resulting interest rate has the required day-counting rule.

7.532.3.14 InterestRate forwardRate (Time t1, Time t2, Compounding comp, Frequency freq = Annual, bool extrapolate = false) const

forward interest rate

returns the implied forward interest rate between two times The resulting interest rate has the same day-counting rule used by the term structure. The same rule should be used for the calculating the time t.

7.532.3.15 Rate parRate (Year tenor, const Date & effectiveDate, Frequency freq = Annual, bool extrapolate = false) const

par rate

returns the implied par rate of a stylised swap starting at the effective date with a given tenor.

Warning:

this par rate is not to be used for evaluation of a real swap, since it does not take into account all the market conventions' details.

7.532.3.16 Rate parRate (Year tenor, Time t0, Frequency freq = Annual, bool extrapolate = false) const

par rate

returns the implied par rate of a stylised swap starting at the given time with a given tenor.

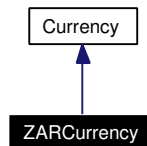
Warning:

this par rate is not to be used for evaluation of a real swap, since it does not take into account all the market conventions' details.

7.533 ZARCurrency Class Reference

```
#include <ql/Currencies/africa.hpp>
```

Inheritance diagram for ZARCurrency:



7.533.1 Detailed Description

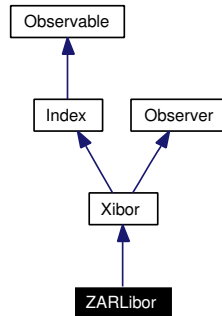
South-African rand.

The ISO three-letter code is ZAR; the numeric code is 710. It is divided into 100 cents.

7.534 ZARLibor Class Reference

```
#include <ql/Indexes/zarlibor.hpp>
```

Inheritance diagram for ZARLibor:



7.534.1 Detailed Description

ZAR Libor index, also known as JIBAR

Todo

check settlement days

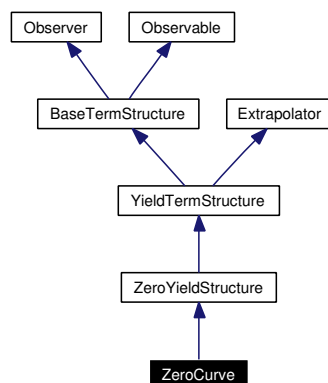
Public Member Functions

- **ZARLibor** ([Integer](#) n, [TimeUnit](#) units, const [Handle](#)< [YieldTermStructure](#) > &h, const [Day-Counter](#) &dc=[Actual365Fixed](#)())

7.535 ZeroCurve Class Reference

```
#include <ql/TermStructures/zerocurve.hpp>
```

Inheritance diagram for ZeroCurve:



7.535.1 Detailed Description

Term structure based on linear interpolation of zero yields.

Public Member Functions

- **ZeroCurve** (const **Date** &todayDate, const std::vector< **Date** > &dates, const std::vector< **Rate** > &yields, const **DayCounter** &dayCounter)
- **ZeroCurve** (const std::vector< **Date** > &dates, const std::vector< **Rate** > &yields, const **DayCounter** &dayCounter)
- **DayCounter** dayCounter () const
the day counter used for date/time conversion
- **Calendar** calendar () const
the calendar used for reference date calculation
- 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) const
zero-yield calculation

7.535.2 Constructor & Destructor Documentation

7.535.2.1 **ZeroCurve** (const **Date** & *today'sDate*, const std::vector< **Date** > & *dates*, const std::vector< **Rate** > & *yields*, const **DayCounter** & *dayCounter*)

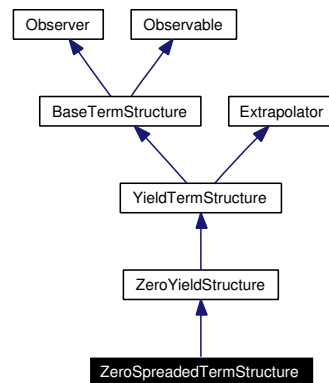
Deprecated

use the constructor without today's date

7.536 ZeroSpreadedTermStructure Class Reference

```
#include <ql/TermStructures/zerospreadedtermstructure.hpp>
```

Inheritance diagram for ZeroSpreadedTermStructure:



7.536.1 Detailed Description

Term structure with an added spread on the zero yield rate.

Note:

This term structure will remain linked to the original structure, i.e., any changes in the latter will be reflected in this structure as well.

Tests

- the correctness of the returned values is tested by checking them against numerical calculations.
- observability against changes in the underlying term structure and in the added spread is checked.

Public Member Functions

- **ZeroSpreadedTermStructure** (const [Handle](#)< [YieldTermStructure](#) > &, const [Handle](#)< [Quote](#) > &spread)

YieldTermStructure interface

- [DayCounter](#) **dayCounter** () const
the day counter used for date/time conversion
- [Calendar](#) **calendar** () const
the calendar used for reference date calculation
- const [Date](#) & **referenceDate** () const
the reference date, i.e., the date at which discount = 1
- const [Date](#) & **todaysDate** () const
today's date

- [Date maxDate \(\)](#) const
the latest date for which the curve can return rates
- [Time maxTime \(\)](#) const
the latest time for which the curve can return rates

Protected Member Functions

- [Rate zeroYieldImpl \(Time\)](#) const
returns the spreaded zero yield rate
- [Rate forwardImpl \(Time\)](#) const
returns the spreaded forward rate

7.536.2 Member Function Documentation

7.536.2.1 [const Date & todaysDate \(\)](#) const [virtual]

today's date

Deprecated

use [Settings::instance\(\).evaluationDate\(\)](#).

Reimplemented from [BaseTermStructure](#).

7.536.2.2 [Rate forwardImpl \(Time\)](#) const [protected, virtual]

returns the spreaded forward rate

Warning:

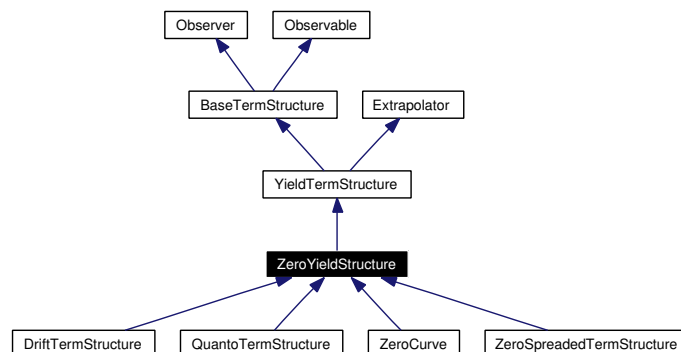
This method must disappear should the spread become a curve

Reimplemented from [ZeroYieldStructure](#).

7.537 ZeroYieldStructure Class Reference

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

Inheritance diagram for ZeroYieldStructure:



7.537.1 Detailed Description

Zero-yield term structure.

This abstract class acts as an adapter to [YieldTermStructure](#) allowing the programmer to implement only the `zeroYieldImpl(Time, bool)` method in derived classes. Discount and forward are calculated from zero yields.

Rates are assumed to be annual continuous compounding.

Public Member Functions

Constructors

See the [BaseTermStructure](#) documentation for issues regarding constructors.

- [ZeroYieldStructure](#) (const [Date](#) &todayDate, const [Date](#) &referenceDate)
- [ZeroYieldStructure](#) ()
- [ZeroYieldStructure](#) (const [Date](#) &referenceDate)
- [ZeroYieldStructure](#) ([Integer](#) settlementDays, const [Calendar](#) &)

Protected Member Functions

YieldTermStructure implementation

- [DiscountFactor](#) `discountImpl (Time)` const
- virtual [Rate](#) `zeroYieldImpl (Time)` const =0
zero-yield calculation
- [Rate](#) `forwardImpl (Time)` const
- [Rate](#) `compoundForwardImpl (Time, Integer)` const

7.537.2 Constructor & Destructor Documentation

7.537.2.1 [ZeroYieldStructure](#) (const [Date](#) & *today'sDate*, const [Date](#) & *referenceDate*)

Deprecated

use the constructor without today's date; set the evaluation date through [Settings::instance\(\)](#).

7.537.3 Member Function Documentation

7.537.3.1 [DiscountFactor](#) discountImpl ([Time](#)) const [protected, virtual]

Returns the discount factor for the given date calculating it from the zero yield.

Implements [YieldTermStructure](#).

7.537.3.2 [Rate](#) forwardImpl ([Time](#)) const [protected, virtual]

Returns the instantaneous forward rate for the given date calculating it from the zero yield.

Implements [YieldTermStructure](#).

Reimplemented in [ZeroSpreadedTermStructure](#).

7.537.3.3 [Rate](#) compoundForwardImpl ([Time](#), [Integer](#)) const [protected, virtual]

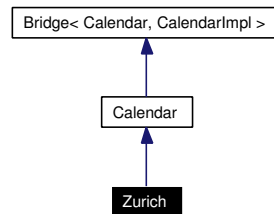
Returns the forward rate at a specified compound frequency for the given date calculating it from the zero yield.

Implements [YieldTermStructure](#).

7.538 Zurich Class Reference

```
#include <ql/Calendars/zurich.hpp>
```

Inheritance diagram for Zurich:



7.538.1 Detailed Description

Zurich calendar

Holidays:

- Saturdays
- Sundays
- New Year's Day, January 1st
- Berchtoldstag, January 2nd
- Good Friday
- Easter Monday
- Ascension Day
- Whit Monday
- Labour Day, May 1st
- National Day, August 1st
- Christmas, December 25th
- St. Stephen's Day, December 26th

Chapter 8

QuantLib File Documentation

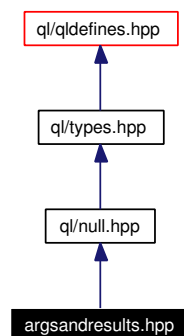
8.1 ql/argsandresults.hpp File Reference

8.1.1 Detailed Description

Base classes for generic arguments and results.

```
#include <ql/null.hpp>
```

Include dependency graph for argsandresults.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Arguments](#)
base class for generic argument groups
- class [Results](#)
base class for generic result groups

Defines

- `#define QL_MIN_VOLATILITY 1.0e-7`
- `#define QL_MIN_DIVYIELD 1.0e-7`
- `#define QL_MAX_VOLATILITY 4.0`
- `#define QL_MAX_DIVYIELD 4.0`

8.2 ql/basetermstructure.hpp File Reference

8.2.1 Detailed Description

base class for term structures

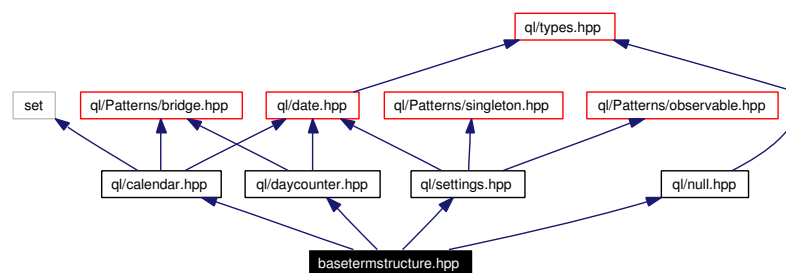
```
#include <ql/calendar.hpp>
```

```
#include <ql/daycounter.hpp>
```

```
#include <ql/settings.hpp>
```

```
#include <ql/null.hpp>
```

Include dependency graph for basetermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BaseTermStructure](#)
Basic term-structure functionality.

8.3 ql/basicdataformatters.hpp File Reference

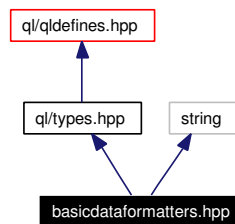
8.3.1 Detailed Description

Classes used to format basic types for output.

```
#include <ql/types.hpp>
```

```
#include <string>
```

Include dependency graph for basicdataformatters.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [IntegerFormatter](#)
Formats integers for output.
- class [SizeFormatter](#)
Formats unsigned integers for output.
- class [DecimalFormatter](#)
Formats real numbers for output.
- class [StringFormatter](#)
Formats strings as lower- or uppercase.
- class [SequenceFormatter](#)
Formats numeric sequences for output.
- class [RateFormatter](#)
Formats rates for output.
- class [VolatilityFormatter](#)
Formats volatilities for output.

8.4 ql/calendar.hpp File Reference

8.4.1 Detailed Description

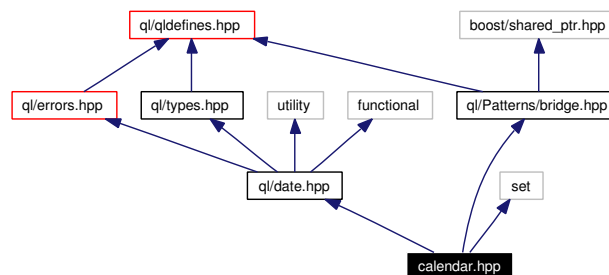
calendar class

```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/bridge.hpp>
```

```
#include <set>
```

Include dependency graph for calendar.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CalendarImpl](#)
abstract base class for calendar implementations
- class [Calendar](#)
calendar class
- class [Calendar::WesternImpl](#)
partial calendar implementation

Enumerations

- enum [BusinessDayConvention](#) {
 [Unadjusted](#), [Preceding](#), [ModifiedPreceding](#), [Following](#),
 [ModifiedFollowing](#), [MonthEndReference](#) }
Business Day conventions.

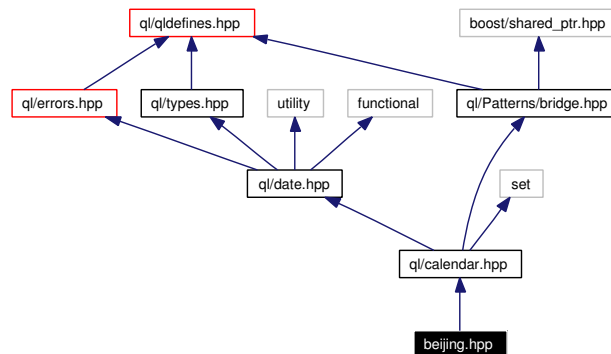
8.5 ql/Calendars/beijing.hpp File Reference

8.5.1 Detailed Description

Beijing calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for beijing.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Beijing](#)
Beijing calendar

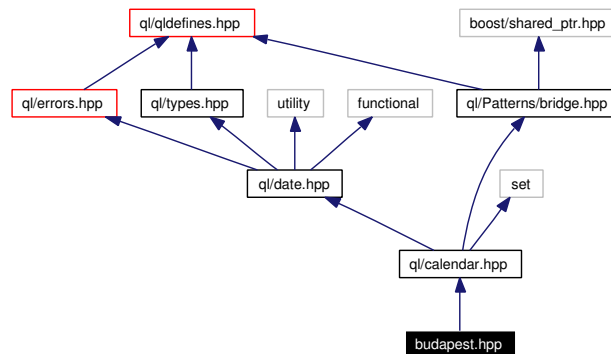
8.6 ql/Calendars/budapest.hpp File Reference

8.6.1 Detailed Description

Budapest calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for budapest.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Budapest**
Budapest calendar

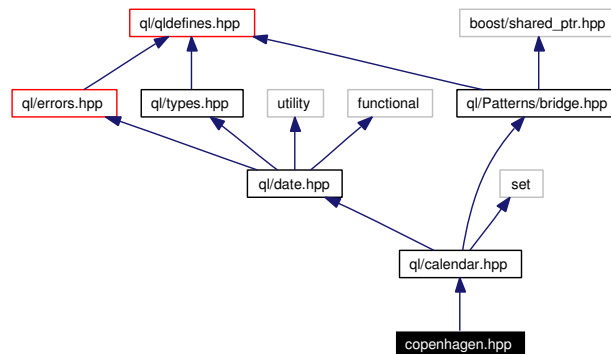
8.7 ql/Calendars/copenhagen.hpp File Reference

8.7.1 Detailed Description

Copenhagen calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for copenhagen.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Copenhagen**
Copenhagen calendar

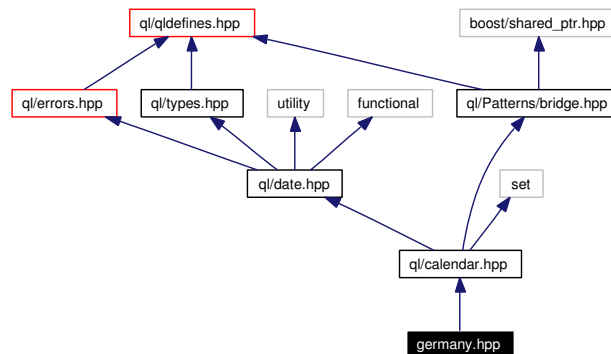
8.8 ql/Calendars/germany.hpp File Reference

8.8.1 Detailed Description

German calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for germany.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Germany](#)
German calendars.

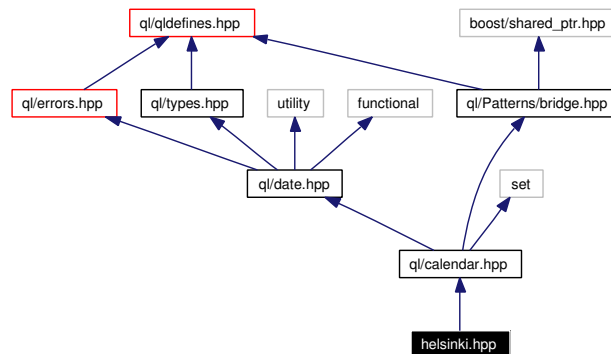
8.9 ql/Calendars/helsinki.hpp File Reference

8.9.1 Detailed Description

Helsinki calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for helsinki.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Helsinki](#)
Helsinki calendar

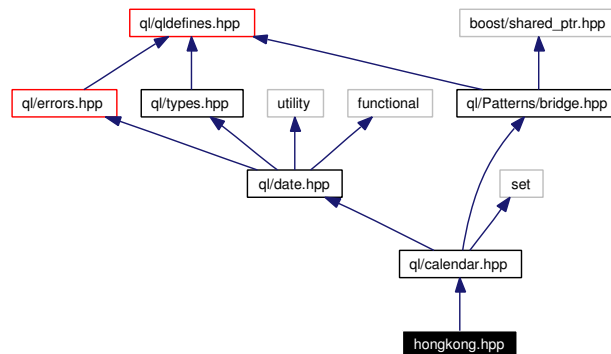
8.10 ql/Calendars/hongkong.hpp File Reference

8.10.1 Detailed Description

Hong Kong calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for hongkong.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [HongKong](#)
Hong Kong calendar.

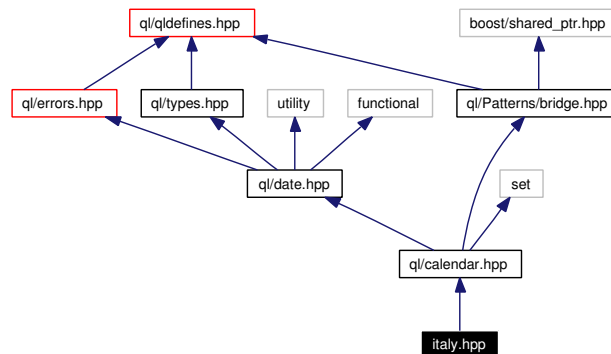
8.11 ql/Calendars/italy.hpp File Reference

8.11.1 Detailed Description

Italian calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for italy.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Italy](#)
Italian calendars.

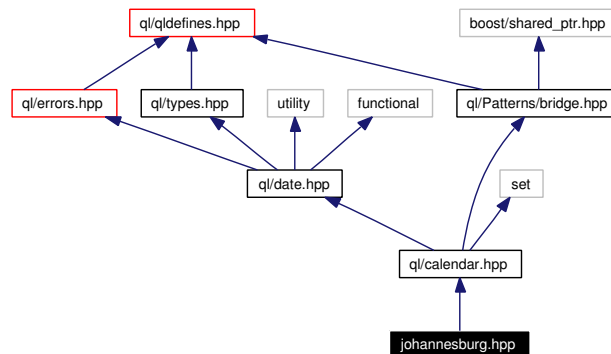
8.12 ql/Calendars/johannesburg.hpp File Reference

8.12.1 Detailed Description

Johannesburg calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for johannesburg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Johannesburg](#)
Johannesburg calendar

8.13 ql/Calendars/jointcalendar.hpp File Reference

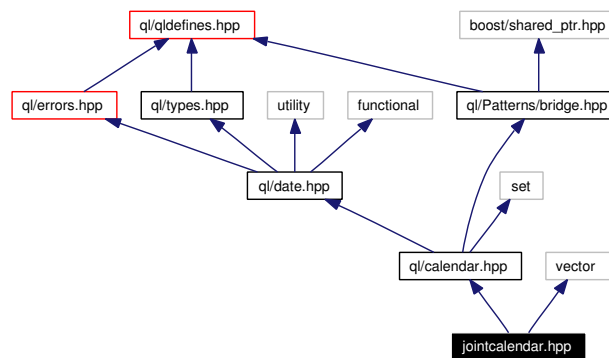
8.13.1 Detailed Description

Joint calendar.

```
#include <ql/calendar.hpp>
```

```
#include <vector>
```

Include dependency graph for jointcalendar.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [JointCalendar](#)
Joint calendar.

Enumerations

- enum **JointCalendarRule** { [JoinHolidays](#), [JoinBusinessDays](#) }
rules for joining calendars

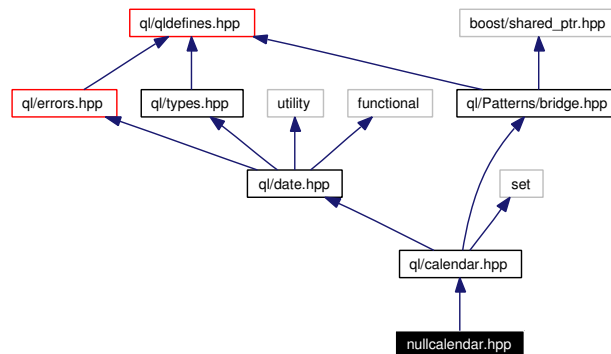
8.14 ql/Calendars/nullcalendar.hpp File Reference

8.14.1 Detailed Description

Calendar for reproducing theoretical calculations.

```
#include <ql/calendar.hpp>
```

Include dependency graph for nullcalendar.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **NullCalendar**
Calendar for reproducing theoretical calculations.

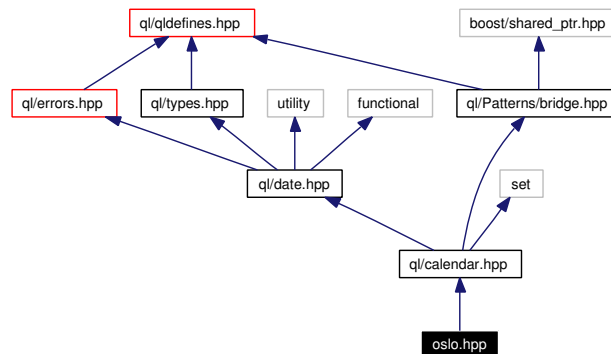
8.15 ql/Calendars/oslo.hpp File Reference

8.15.1 Detailed Description

Oslo calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for oslo.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Oslo](#)
Oslo calendar

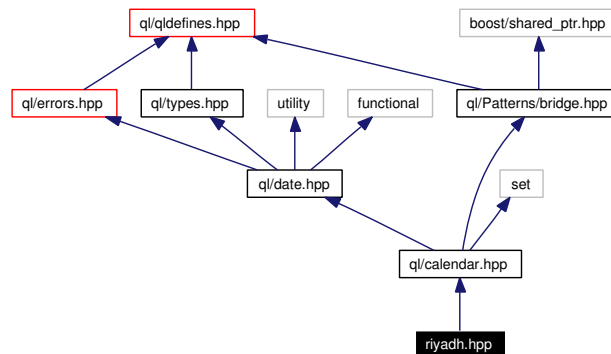
8.16 ql/Calendars/riyadh.hpp File Reference

8.16.1 Detailed Description

Riyadh calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for riyadh.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Riyadh](#)
Riyadh calendar

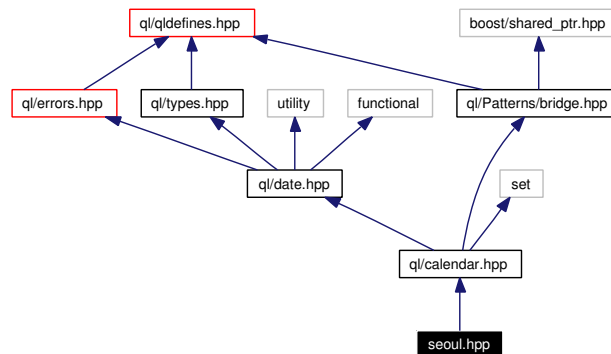
8.17 ql/Calendars/seoul.hpp File Reference

8.17.1 Detailed Description

South Korea calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for seoul.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Seoul](#)
Seoul calendar

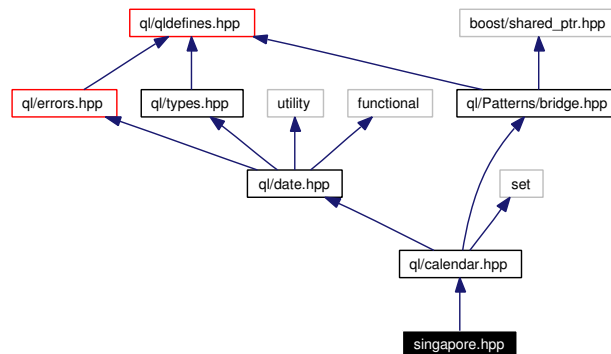
8.18 ql/Calendars/singapore.hpp File Reference

8.18.1 Detailed Description

Singapore calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for singapore.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Singapore](#)
Singapore calendar

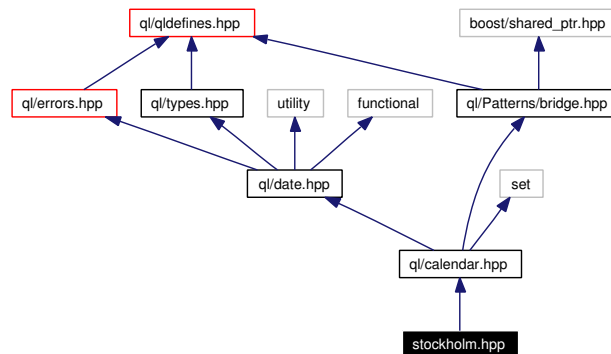
8.19 ql/Calendars/stockholm.hpp File Reference

8.19.1 Detailed Description

Stockholm calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for stockholm.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Stockholm](#)
Stockholm calendar

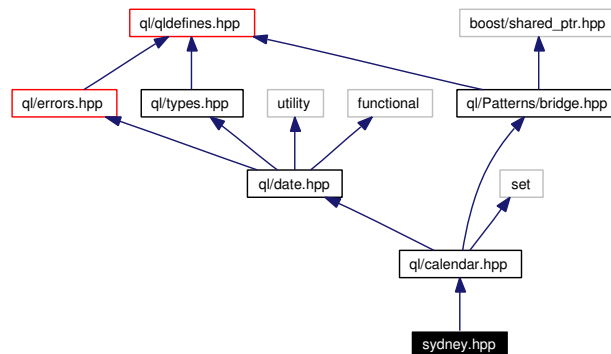
8.20 ql/Calendars/sydney.hpp File Reference

8.20.1 Detailed Description

Sydney calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for sydney.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Sydney](#)
Sydney calendar (New South Wales, Australia)

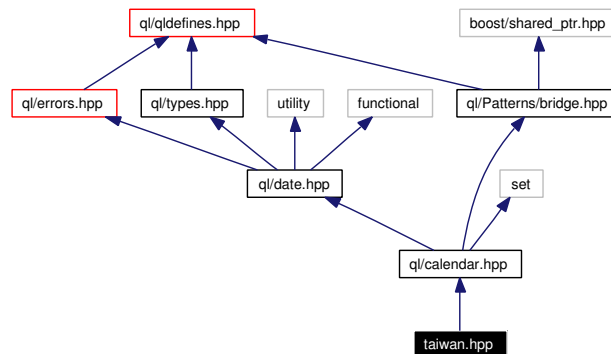
8.21 ql/Calendars/taiwan.hpp File Reference

8.21.1 Detailed Description

Taiwan calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for taiwan.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Taiwan](#)
Taiwan calendar

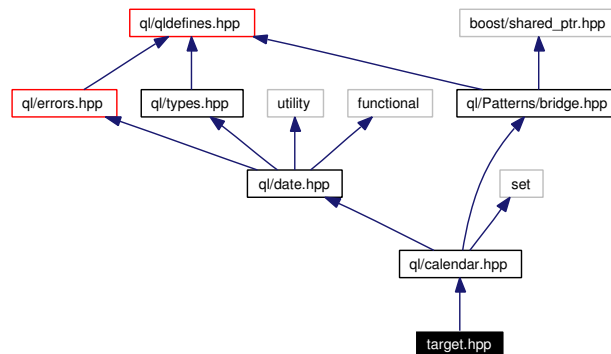
8.22 ql/Calendars/target.hpp File Reference

8.22.1 Detailed Description

TARGET calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for target.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TARGET**
TARGET calendar

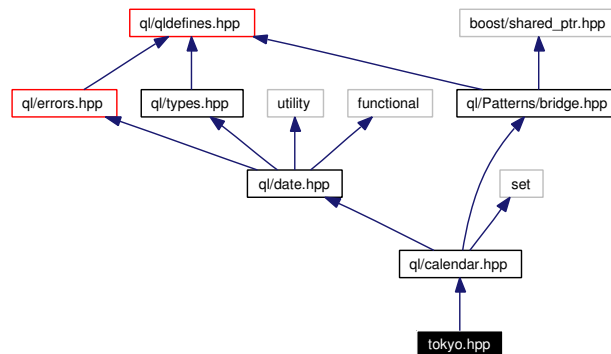
8.23 ql/Calendars/tokyo.hpp File Reference

8.23.1 Detailed Description

Tokyo calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for tokyo.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Tokyo**
Tokyo calendar

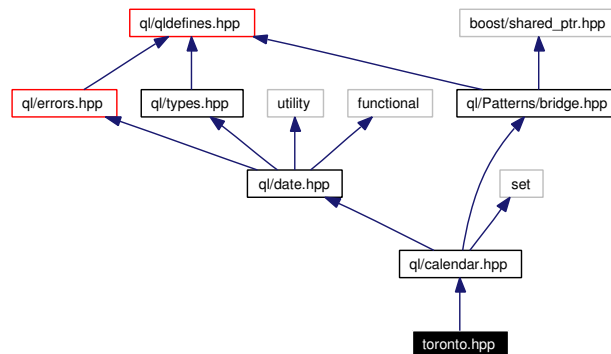
8.24 ql/Calendars/toronto.hpp File Reference

8.24.1 Detailed Description

Toronto calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for `toronto.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [Toronto](#)
Toronto calendar

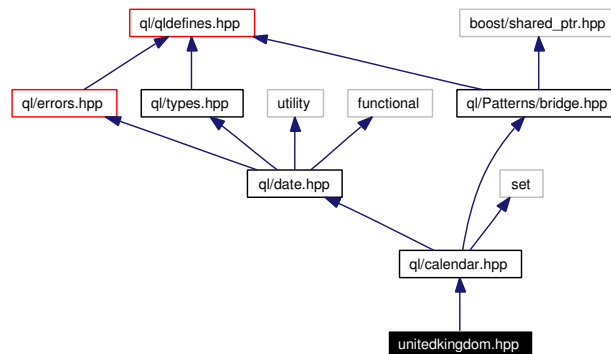
8.25 ql/Calendars/unitedkingdom.hpp File Reference

8.25.1 Detailed Description

UK calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for unitedkingdom.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **UnitedKingdom**
United Kingdom calendars.

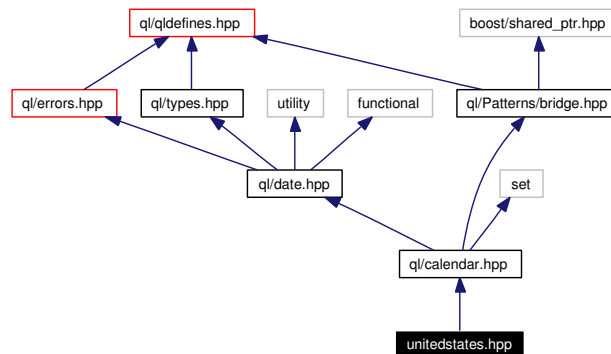
8.26 ql/Calendars/unitedstates.hpp File Reference

8.26.1 Detailed Description

US calendars.

```
#include <ql/calendar.hpp>
```

Include dependency graph for unitedstates.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **UnitedStates**
United States calendars.

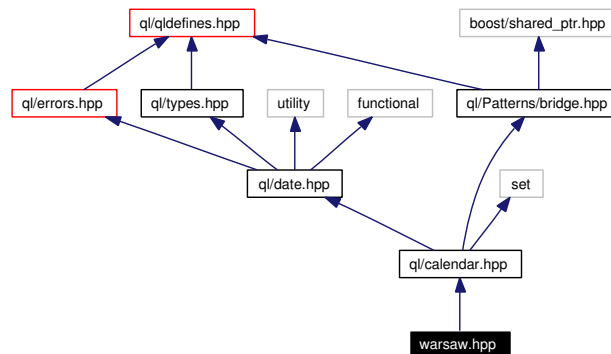
8.27 ql/Calendars/warsaw.hpp File Reference

8.27.1 Detailed Description

Warsaw calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for warsaw.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Warsaw](#)
Warsaw calendar

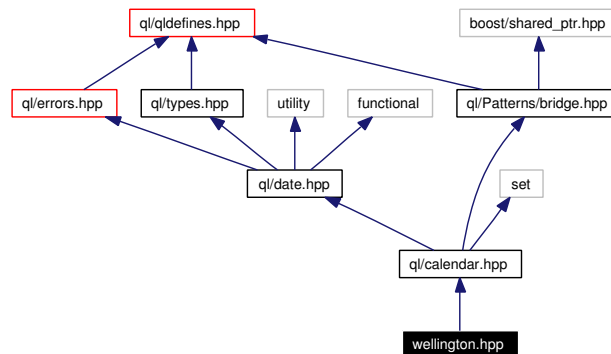
8.28 ql/Calendars/wellington.hpp File Reference

8.28.1 Detailed Description

Wellington calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for wellington.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Wellington](#)
Wellington calendar

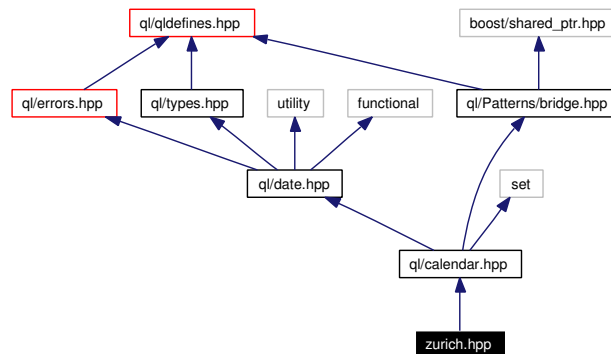
8.29 ql/Calendars/zurich.hpp File Reference

8.29.1 Detailed Description

Zurich calendar.

```
#include <ql/calendar.hpp>
```

Include dependency graph for zurich.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Zurich](#)
Zurich calendar

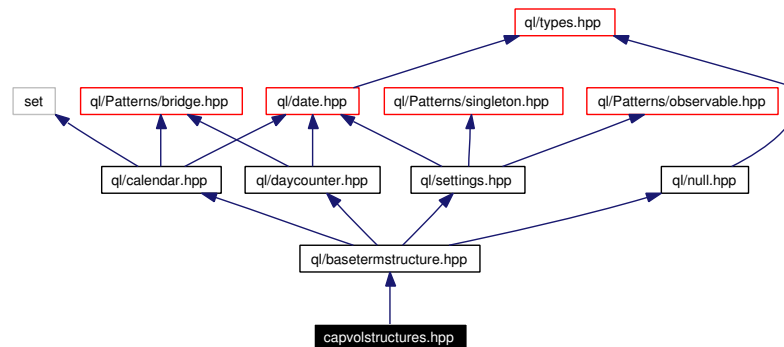
8.30 ql/capvolstructures.hpp File Reference

8.30.1 Detailed Description

Cap/Floor volatility structures.

```
#include <ql/basetermstructure.hpp>
```

Include dependency graph for capvolstructures.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CapVolatilityStructure](#)
Cap/floor term-volatility structure.
- class [CapletVolatilityStructure](#)
Caplet/floorlet forward-volatility structure.

Typedefs

- typedef [CapVolatilityStructure](#) [CapFlatVolatilityStructure](#)
- typedef [CapletVolatilityStructure](#) [CapletForwardVolatilityStructure](#)

8.30.2 Typedef Documentation

8.30.2.1 typedef [CapVolatilityStructure](#) [CapFlatVolatilityStructure](#)

Deprecated

renamed to [CapVolatilityStructure](#)

8.30.2.2 `typedef CapletVolatilityStructure CapletForwardVolatilityStructure`

Deprecated

renamed to [CapletVolatilityStructure](#)

8.31 ql/cashflow.hpp File Reference

8.31.1 Detailed Description

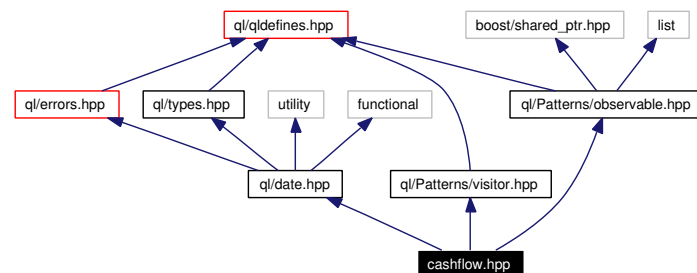
Base class for cash flows.

```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

```
#include <ql/Patterns/visitor.hpp>
```

Include dependency graph for cashflow.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CashFlow**
Base class for cash flows.

8.32 ql/CashFlows/basispointsensitivity.hpp File Reference

8.32.1 Detailed Description

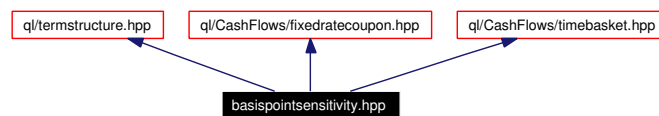
basis point sensitivity calculator

```
#include <ql/termstructure.hpp>
```

```
#include <ql/CashFlows/fixedratecoupon.hpp>
```

```
#include <ql/CashFlows/timebasket.hpp>
```

Include dependency graph for basispointsensitivity.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BPSCalculator**
basis point sensitivity (BPS) calculator
- class **BPSBasketCalculator**

Functions

- **Real BasisPointSensitivity** (const std::vector< boost::shared_ptr< CashFlow > > &, const Handle< YieldTermStructure > &)
Collective basis-point sensitivity of a cash-flow sequence.
- TimeBasket **BasisPointSensitivityBasket** (const std::vector< boost::shared_ptr< CashFlow > > &, const Handle< YieldTermStructure > &, **Integer** basis)

8.32.2 Function Documentation

8.32.2.1 TimeBasket BasisPointSensitivityBasket (const std::vector< boost::shared_ptr< CashFlow > > &, const Handle< YieldTermStructure > &, **Integer** basis)

Bug

This function must still be checked. It is not guaranteed to yield the right results.

8.33 ql/CashFlows/cashflowvectors.hpp File Reference

8.33.1 Detailed Description

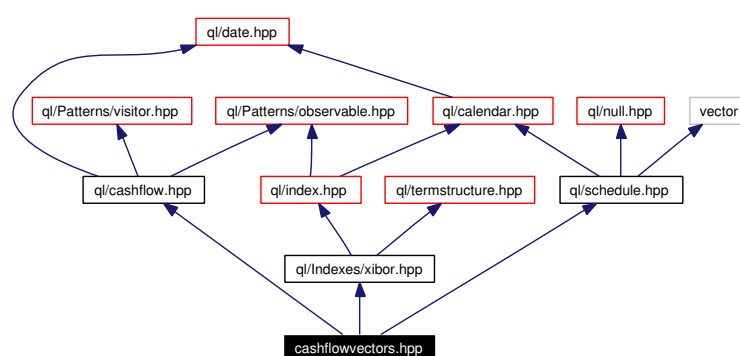
Cash flow vector builders.

```
#include <ql/cashflow.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/schedule.hpp>
```

Include dependency graph for cashflowvectors.hpp:



Namespaces

- namespace **QuantLib**

Functions

- `std::vector< boost::shared_ptr< CashFlow > >` [FixedRateCouponVector](#) (const Schedule &schedule, [BusinessDayConvention](#) paymentAdjustment, const std::vector< [Real](#) > &nominals, const std::vector< [Rate](#) > &couponRates, const DayCounter &dayCount, const DayCounter &firstPeriodDayCount=DayCounter())

helper function building a sequence of fixed rate coupons

- `std::vector< boost::shared_ptr< CashFlow > >` [FloatingRateCouponVector](#) (const Schedule &schedule, [BusinessDayConvention](#) paymentAdjustment, const std::vector< [Real](#) > &nominals, const boost::shared_ptr< Xibor > &index, [Integer](#) fixingDays, const std::vector< [Spread](#) > &spreads=std::vector< [Spread](#) >(), const DayCounter &dayCounter=DayCounter())

helper function building a sequence of par coupons

8.33.2 Function Documentation

8.33.2.1 `std::vector<boost::shared_ptr<CashFlow> > FloatingRateCouponVector (const Schedule & schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > & nominals, const boost::shared_ptr< Xibor > & index, Integer fixingDays, const std::vector< Spread > & spreads = std::vector< Spread >(), const DayCounter & dayCounter = DayCounter())`

helper function building a sequence of par coupons

Either UpFrontIndexedCoupons or ParCoupons are used depending on the library configuration.

[Todo](#)

A suitable algorithm should be implemented for the calculation of the interpolated index fixing for a short/long first coupon.

8.34 ql/CashFlows/coupon.hpp File Reference

8.34.1 Detailed Description

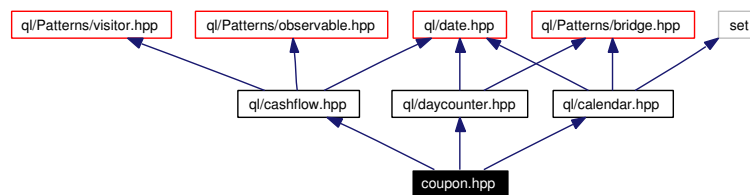
Coupon accruing over a fixed period.

```
#include <ql/cashflow.hpp>
```

```
#include <ql/calendar.hpp>
```

```
#include <ql/daycounter.hpp>
```

Include dependency graph for coupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Coupon**
coupon accruing over a fixed period

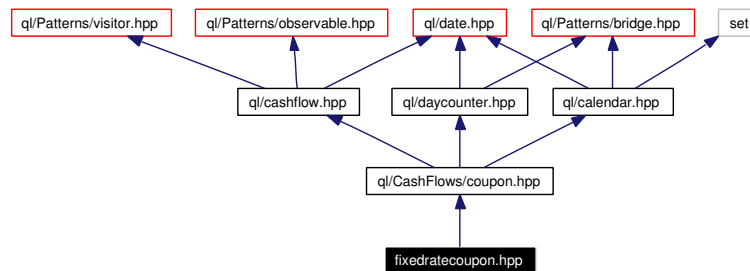
8.35 ql/CashFlows/fixedratecoupon.hpp File Reference

8.35.1 Detailed Description

Coupon paying a fixed annual rate.

```
#include <ql/CashFlows/coupon.hpp>
```

Include dependency graph for fixedratecoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FixedRateCoupon](#)
Coupon paying a fixed interest rate

8.36 ql/CashFlows/floatingratecoupon.hpp File Reference

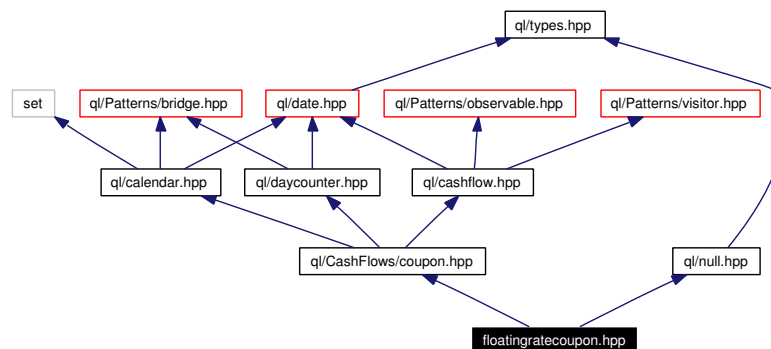
8.36.1 Detailed Description

Coupon paying a variable rate.

```
#include <ql/CashFlows/coupon.hpp>
```

```
#include <ql/null.hpp>
```

Include dependency graph for floatingratecoupon.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `FloatingRateCoupon`
Coupon paying a variable rate

8.37 ql/CashFlows/inarrearindexedcoupon.hpp File Reference

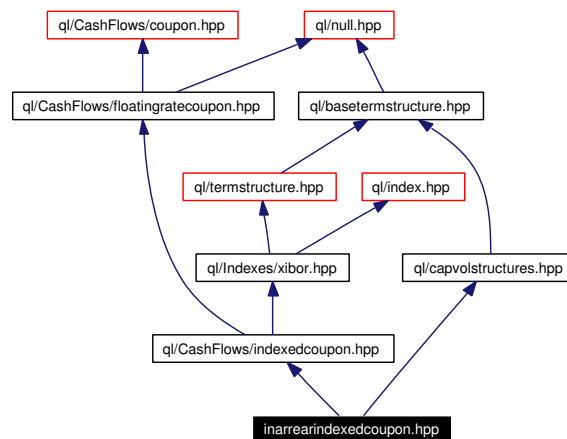
8.37.1 Detailed Description

in-arrear floating-rate coupon

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

```
#include <ql/capvolstructures.hpp>
```

Include dependency graph for inarrearindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InArrearIndexedCoupon](#)
In-arrear floating-rate coupon.

8.38 ql/CashFlows/indexcashflowvectors.hpp File Reference

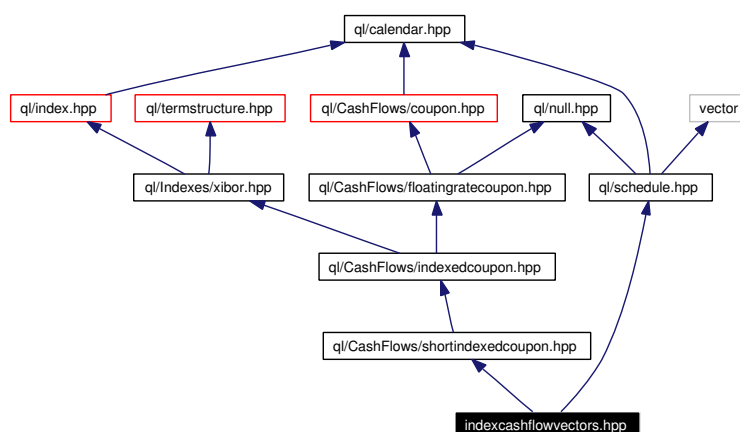
8.38.1 Detailed Description

Index Cash flow vector builders.

```
#include <ql/CashFlows/shortindexedcoupon.hpp>
```

```
#include <ql/schedule.hpp>
```

Include dependency graph for indexcashflowvectors.hpp:



Namespaces

- namespace **QuantLib**

Functions

- `template<class IndexedCouponType> std::vector< boost::shared_ptr< CashFlow > > IndexedCouponVector (const Schedule &schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > &nominals, const boost::shared_ptr< Xibor > &index, Integer fixingDays, const std::vector< Spread > &spreads, const DayCounter &dayCounter=DayCounter())`

helper function building a leg of floating coupons

8.38.2 Function Documentation

- 8.38.2.1 `std::vector<boost::shared_ptr<CashFlow> > IndexedCouponVector (const Schedule & schedule, BusinessDayConvention paymentAdjustment, const std::vector< Real > & nominals, const boost::shared_ptr< Xibor > & index, Integer fixingDays, const std::vector< Spread > & spreads, const DayCounter & dayCounter = DayCounter())`

helper function building a leg of floating coupons

Either [ParCoupon](#), [UpFrontIndexedCoupon](#), [InArrearIndexedCoupon](#), or any other coupon can be used whose constructor takes the same arguments.

Warning:

The last argument is used due to a known VC++ bug regarding function template instantiation. It must be passed explicitly when using the function with that compiler; the simplest choice for the value to be passed is `(const Type*) 0` where `Type` is the desired coupon type.

8.39 ql/CashFlows/indexedcoupon.hpp File Reference

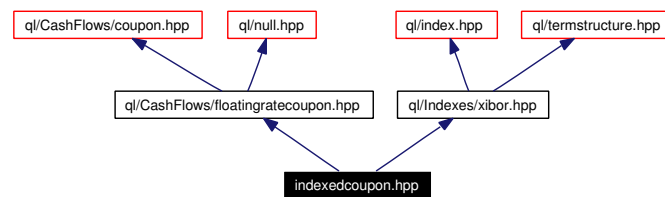
8.39.1 Detailed Description

indexed coupon

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for indexedcoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [IndexedCoupon](#)
Base indexed coupon class.

8.40 ql/CashFlows/parcoupon.hpp File Reference

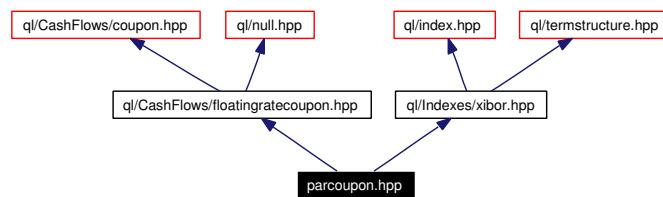
8.40.1 Detailed Description

Coupon at par on a term structure.

```
#include <ql/CashFlows/floatingratecoupon.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for parcoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ParCoupon](#)
coupon at par on a term structure

8.41 ql/CashFlows/shortfloatingcoupon.hpp File Reference

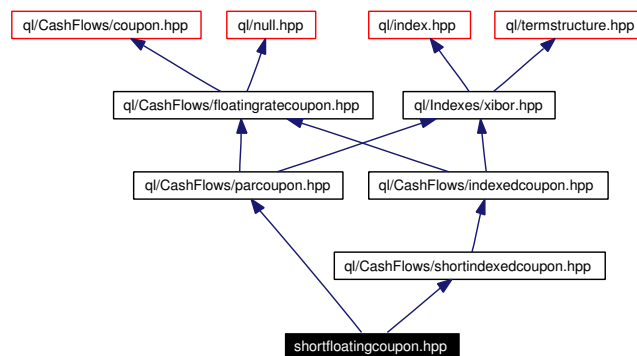
8.41.1 Detailed Description

Short (or long) coupon at par on a term structure.

```
#include <ql/CashFlows/parcoupon.hpp>
```

```
#include <ql/CashFlows/shortindexedcoupon.hpp>
```

Include dependency graph for shortfloatingcoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Short< ParCoupon >](#)
Short coupon at par on a term structure

Typedefs

- typedef `Short< ParCoupon >` [ShortFloatingRateCoupon](#)

8.41.2 Typedef Documentation

8.41.2.1 typedef `Short<ParCoupon>` `ShortFloatingRateCoupon`

Deprecated

use [Short<ParCoupon>](#) instead

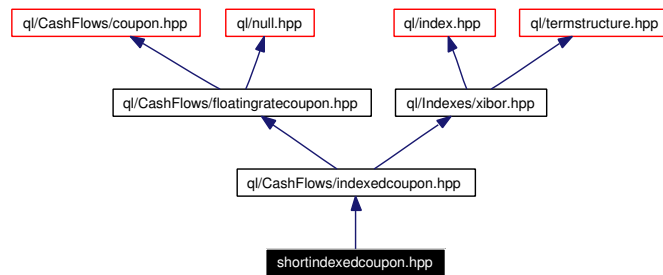
8.42 ql/CashFlows/shortindexedcoupon.hpp File Reference

8.42.1 Detailed Description

Short (or long) indexed coupon.

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

Include dependency graph for shortindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Short](#)
Short indexed coupon

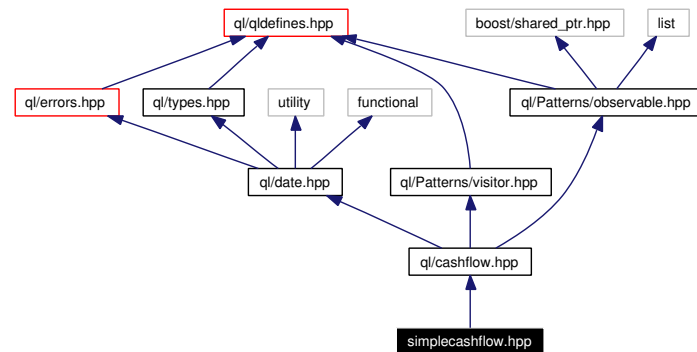
8.43 ql/CashFlows/simplecashflow.hpp File Reference

8.43.1 Detailed Description

Predetermined cash flow.

```
#include <ql/cashflow.hpp>
```

Include dependency graph for simplecashflow.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SimpleCashFlow](#)
Predetermined cash flow.

8.44 ql/CashFlows/timebasket.hpp File Reference

8.44.1 Detailed Description

Distribution over a number of dates

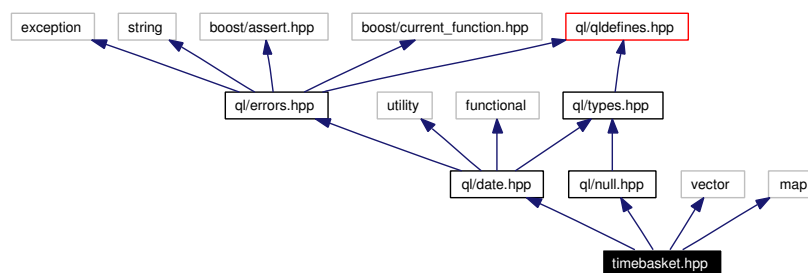
```
#include <ql/date.hpp>
```

```
#include <ql/null.hpp>
```

```
#include <vector>
```

```
#include <map>
```

Include dependency graph for timebasket.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TimeBasket](#)

Distribution over a number of dates.

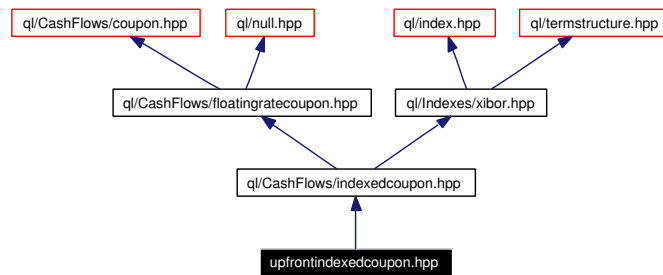
8.45 ql/CashFlows/upfrontindexedcoupon.hpp File Reference

8.45.1 Detailed Description

Up front indexed coupon.

```
#include <ql/CashFlows/indexedcoupon.hpp>
```

Include dependency graph for upfrontindexedcoupon.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **UpFrontIndexedCoupon**
up front indexed coupon class

8.46 ql/Currencies/africa.hpp File Reference

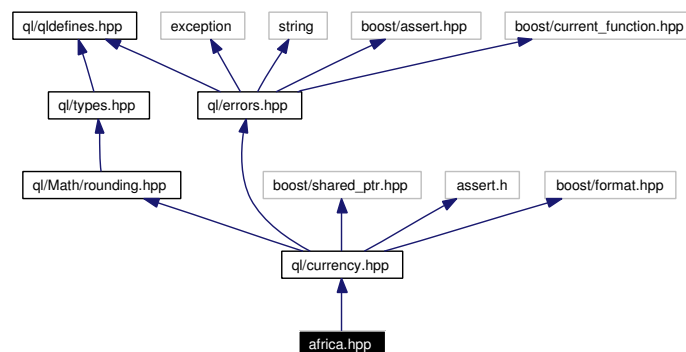
8.46.1 Detailed Description

African currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

```
#include <ql/currency.hpp>
```

Include dependency graph for africa.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ZARCurrency](#)
South-African rand.

8.47 ql/Currencies/america.hpp File Reference

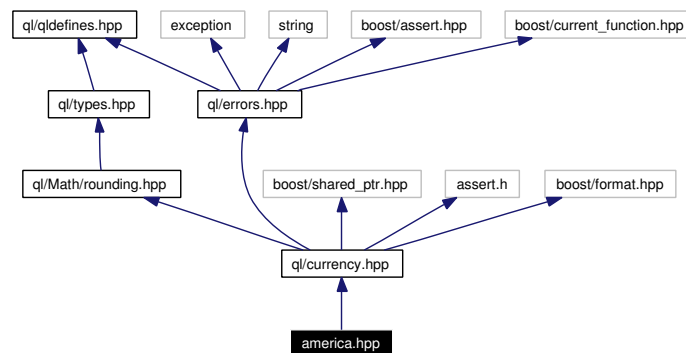
8.47.1 Detailed Description

American currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

```
#include <ql/currency.hpp>
```

Include dependency graph for america.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ARSCurrency](#)
Argentinian peso.
- class [BRLCurrency](#)
Brazilian real.
- class [CADCurrency](#)
Canadian dollar.
- class [CLPCurrency](#)
Chilean peso.
- class [COPCurrency](#)
Colombian peso.
- class [MXNCurrency](#)
Mexican peso.
- class [TTDCurrency](#)

Trinidad & Tobago dollar.

- class [USDCurrency](#)
U.S. dollar.
- class [VEBCurrency](#)
Venezuelan bolivar.

8.48 ql/Currencies/asia.hpp File Reference

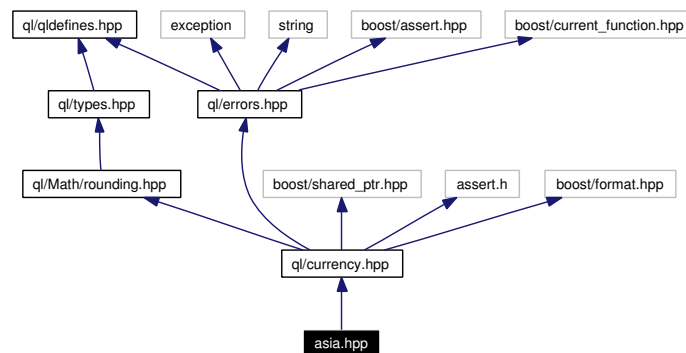
8.48.1 Detailed Description

Asian currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

```
#include <ql/currency.hpp>
```

Include dependency graph for asia.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BDTCurrency**
Bangladesh taka.
- class **CNYCurrency**
Chinese yuan.
- class **HKDCurrency**
Honk Kong dollar.
- class **ILSCurrency**
Israeli shekel.
- class **INRCurrency**
Indian rupee.
- class **IQDCurrency**
Iraqi dinar.
- class **IRRCurrency**

Iranian rial.

- class [JPYCurrency](#)
Japanese yen.
- class [KRWCurrency](#)
South-Korean won.
- class [KWDCurrency](#)
Kuwaiti dinar.
- class [NPRCurrency](#)
Nepal rupee.
- class [PKRCurrency](#)
Pakistani rupee.
- class [SARCurrency](#)
Saudi riyal.
- class [SGDCurrency](#)
Singapore dollar.
- class [THBCurrency](#)
Thai baht.
- class [TWDCurrency](#)
Taiwan dollar.

8.49 ql/Currencies/europe.hpp File Reference

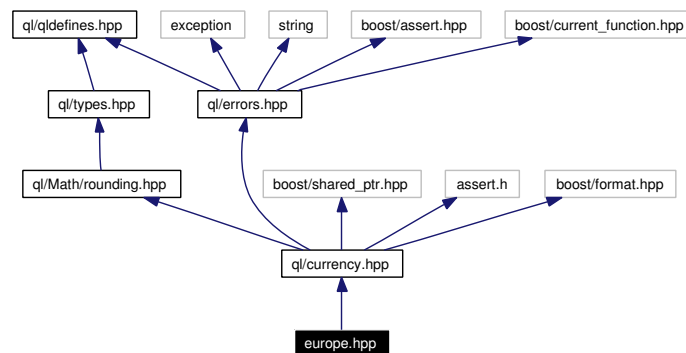
8.49.1 Detailed Description

European currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

```
#include <ql/currency.hpp>
```

Include dependency graph for europe.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BGLCurrency**
Bulgarian lev.
- class **BYRCurrency**
Belarussian ruble.
- class **CHFCurrency**
Swiss franc.
- class **CYPCurrency**
Cyprus pound.
- class **CZKCurrency**
Czech koruna.
- class **DKKCurrency**
Danish krone.
- class **EEKCurrency**

Estonian kroon.

- class [EURCurrency](#)

European Euro.

- class [GBPCurrency](#)

British pound sterling.

- class [HUFCurrency](#)

Hungarian forint.

- class [ISKCurrency](#)

Iceland krona.

- class [LTLCurrency](#)

Lithuanian litas.

- class [LVLCurrency](#)

Latvian lat.

- class [MTLCurrency](#)

Maltese lira.

- class [NOKCurrency](#)

Norwegian krone.

- class [PLNCurrency](#)

Polish zloty.

- class [ROLCurrency](#)

Romanian leu.

- class [SEKCurrency](#)

Swedish krona.

- class [SITCurrency](#)

Slovenian tolar.

- class [SKKCurrency](#)

Slovak koruna.

- class [TRLCurrency](#)

Turkish lira.

- class [ATSCurrency](#)

Austrian shilling.

- class [BEFCurrency](#)

Belgian franc.

- class [DEMCurrency](#)
Deutsche mark.
- class [ESPCurrency](#)
Spanish peseta.
- class [FIMCurrency](#)
Finnish markka.
- class [FRFCurrency](#)
French franc.
- class [GRDCurrency](#)
Greek drachma.
- class [IEPCurrency](#)
Irish punt.
- class [ITLCurrency](#)
Italian lira.
- class [LUFCurrency](#)
Luxembourg franc.
- class [NLGCurrency](#)
Dutch guilder.
- class [PTECurrency](#)
Portuguese escudo.

8.50 ql/Currencies/exchangeratemanager.hpp File Reference

8.50.1 Detailed Description

exchange-rate repository

```
#include <ql/exchangerate.hpp>
```

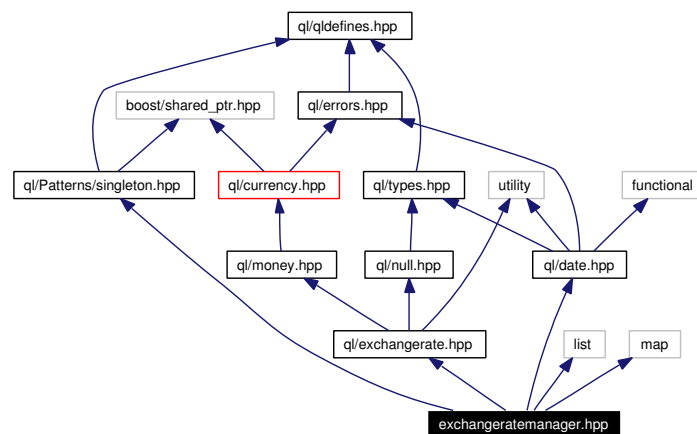
```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <list>
```

```
#include <map>
```

Include dependency graph for `exchangeratemanager.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [ExchangeRateManager](#)
exchange-rate repository

8.51 ql/Currencies/oceania.hpp File Reference

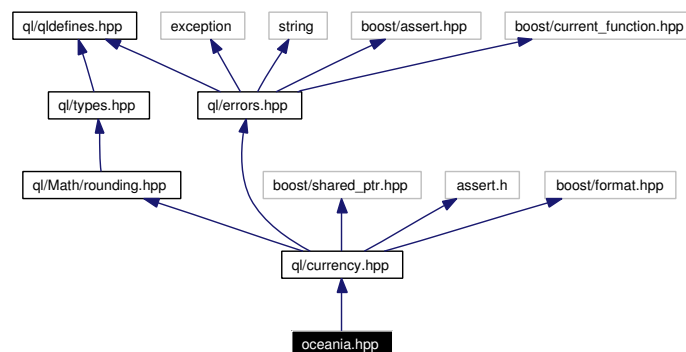
8.51.1 Detailed Description

Oceanian currencies.

Data from http://fx.sauder.ubc.ca/currency_table.html and <http://www.thefinancials.com/vortex/CurrencyFormats.html>

```
#include <ql/currency.hpp>
```

Include dependency graph for oceania.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **AUDCurrency**
Australian dollar.
- class **NZDCurrency**
New Zealand dollar.

8.52 ql/currency.hpp File Reference

8.52.1 Detailed Description

Known currencies.

```
#include <ql/Math/rounding.hpp>
```

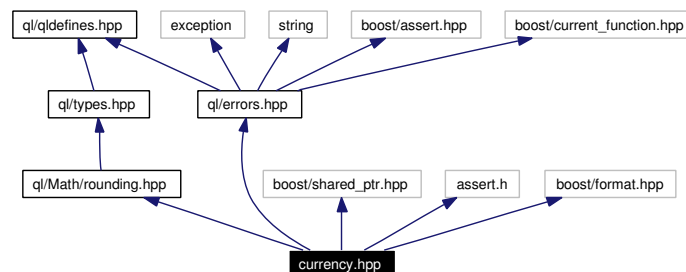
```
#include <ql/errors.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

```
#include <assert.h>
```

```
#include <boost/format.hpp>
```

Include dependency graph for currency.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Currency](#)
Currency specification
- class [CurrencyFormatter](#)
format currencies for output

Enumerations

- enum [CurrencyTag](#) {
[ARS](#), [ATS](#), [AUD](#), [BDT](#),
[BEF](#), [BGL](#), [BRL](#), [BYB](#),
[CAD](#), [CHF](#), [CLP](#), [CNY](#),
[COP](#), [CYP](#), [CZK](#), [DEM](#),
[DKK](#), [EEK](#), [EUR](#), [GBP](#),
[GRD](#), [HKD](#), [HUF](#), [ILS](#),
[INR](#), [IQD](#), [IRR](#), [ISK](#),

ITL, JPY, KRW, KWD,
LTL, LVL, MTL, MXP,
NOK, NPR, NZD, PKR,
PLN, ROL, SAR, SEK,
SGD, SIT, SKK, THB,
TRL, TTD, TWD, USD,
VEB, ZAR }

Tags for known currencies.

Variables

- Currency [make_currency](#) ([CurrencyTag](#))
Converts currency tags to [Currency](#) instances.

8.52.2 Variable Documentation

8.52.2.1 Currency [make_currency](#)([CurrencyTag](#))

Converts currency tags to [Currency](#) instances.

Deprecated

to be used while migrating away from [CurrencyTag](#). Use [Currency](#) directly.

8.53 ql/dataformatters.hpp File Reference

8.53.1 Detailed Description

Classes used to format data for output.

```
#include <ql/basicdataformatters.hpp>
```

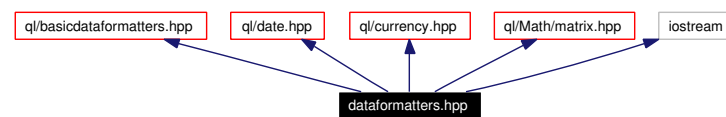
```
#include <ql/date.hpp>
```

```
#include <ql/currency.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

```
#include <iostream>
```

Include dependency graph for dataformatters.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [EuroFormatter](#)
Formats amounts in Euro for output.

8.54 ql/dataparsers.hpp File Reference

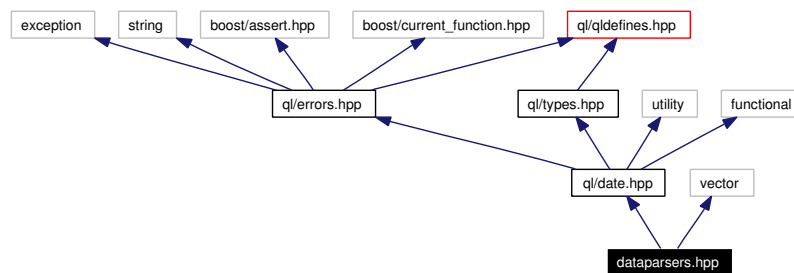
8.54.1 Detailed Description

Classes used to parse data for input.

```
#include <ql/date.hpp>
```

```
#include <vector>
```

Include dependency graph for dataparsers.hpp:



Namespaces

- namespace **QuantLib**

8.55 ql/date.hpp File Reference

8.55.1 Detailed Description

date- and time-related classes, typedefs and enumerations

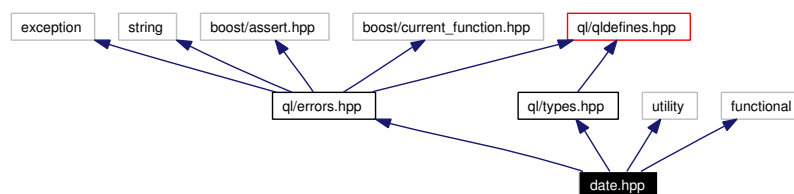
```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <utility>
```

```
#include <functional>
```

Include dependency graph for date.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Period**
Time period described by a number of a given time unit.
- class **Date**
Concrete date class.
- class **DateFormatter**
Formats dates for output.
- class **WeekdayFormatter**
Formats weekday for output.
- class **FrequencyFormatter**
Formats frequency for output.

Typedefs

- typedef **Integer Day**
Day number.
- typedef **Integer Year**

Year number.

Enumerations

- enum [Weekday](#) {
 Sunday = 1, **Monday** = 2, **Tuesday** = 3, **Wednesday** = 4,
 Thursday = 5, **Friday** = 6, **Saturday** = 7 }
- enum [Month](#) {
 January = 1, **February** = 2, **March** = 3, **April** = 4,
 May = 5, **June** = 6, **July** = 7, **August** = 8,
 September = 9, **October** = 10, **November** = 11, **December** = 12 }
 Month names.
- enum [IMMMonth](#) { **H** = 3, **M** = 6, **U** = 9, **Z** = 12 }
 Main cycle of the International Money Market (a.k.a. IMM) Months.
- enum [Frequency](#) {
 NoFrequency = -1, **Once** = 0, **Annual** = 1, **Semiannual** = 2,
 EveryFourthMonth = 3, **Quarterly** = 4, **Bimonthly** = 6, **Monthly** = 12 }
 Frequency of events.
- enum [TimeUnit](#) { **Days**, **Weeks**, **Months**, **Years** }
 Units used to describe time periods.

8.56 ql/daycounter.hpp File Reference

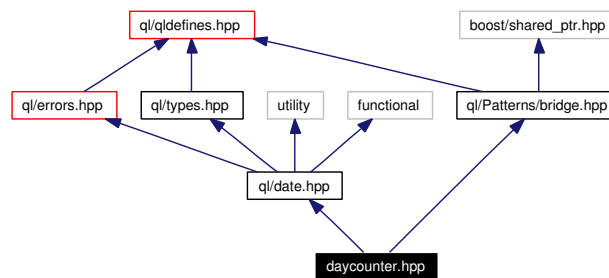
8.56.1 Detailed Description

day counter class

```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/bridge.hpp>
```

Include dependency graph for daycounter.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DayCounterImpl](#)
abstract base class for day counter implementations
- class [DayCounter](#)
day counter class

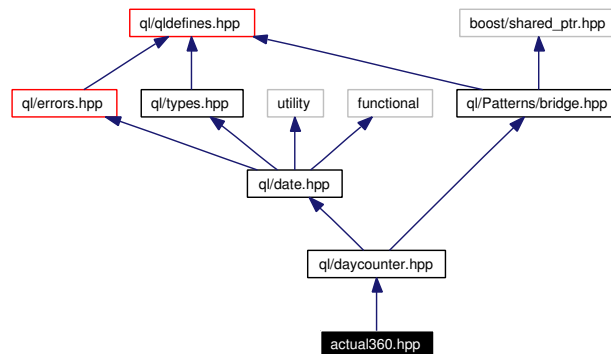
8.57 ql/DayCounters/actual360.hpp File Reference

8.57.1 Detailed Description

act/360 day counter

```
#include <ql/daycounter.hpp>
```

Include dependency graph for actual360.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Actual360](#)
Actual/360 day count convention.

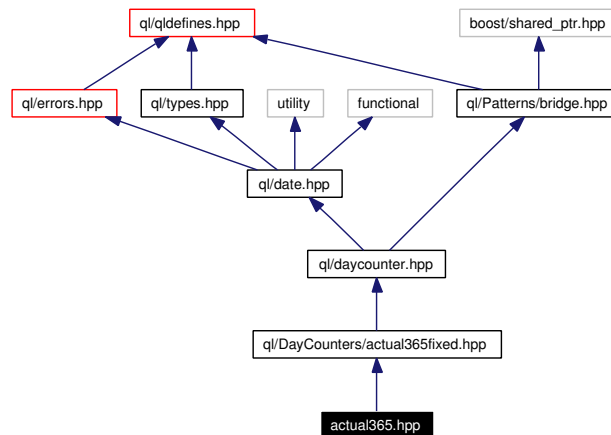
8.58 ql/DayCounters/actual365.hpp File Reference

8.58.1 Detailed Description

act/365 day counter

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for actual365.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef Actual365Fixed [Actual365](#)

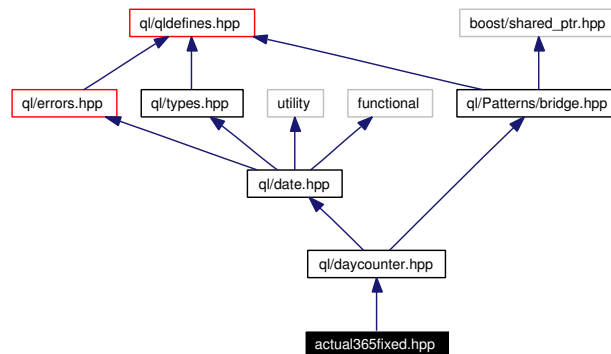
8.59 ql/DayCounters/actual365fixed.hpp File Reference

8.59.1 Detailed Description

Actual/365 (Fixed) day counter.

```
#include <ql/daycounter.hpp>
```

Include dependency graph for actual365fixed.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Actual365Fixed](#)
Actual/365 (Fixed) day count convention.

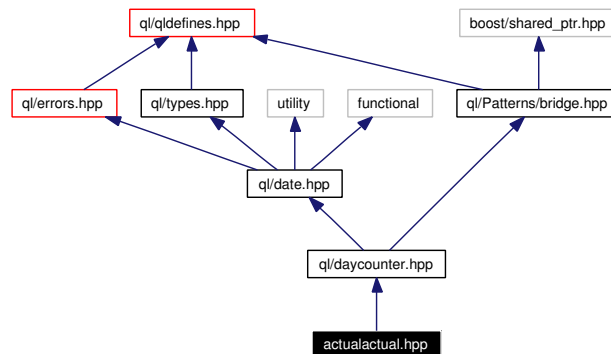
8.60 ql/DayCounters/actualactual.hpp File Reference

8.60.1 Detailed Description

act/act day counters

```
#include <ql/daycounter.hpp>
```

Include dependency graph for actualactual.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ActualActual](#)
Actual/Actual day count.

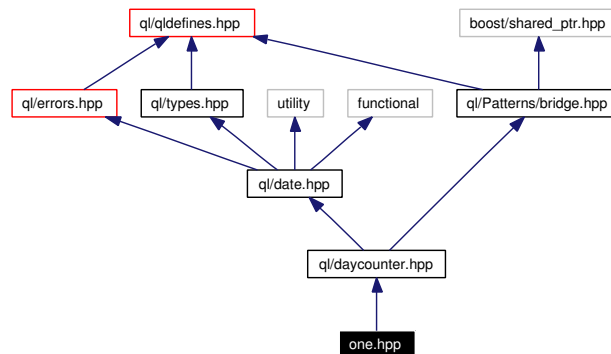
8.61 ql/DayCounters/one.hpp File Reference

8.61.1 Detailed Description

1/1 day counter

```
#include <ql/daycounter.hpp>
```

Include dependency graph for one.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OneDayCounter](#)
1/1 day count convention

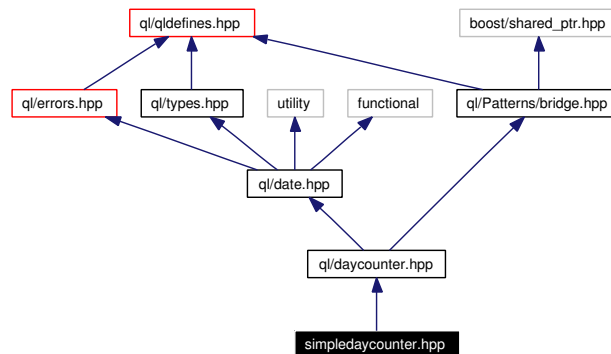
8.62 ql/DayCounters/simpliedaycounter.hpp File Reference

8.62.1 Detailed Description

Simple day counter for reproducing theoretical calculations.

```
#include <ql/daycounter.hpp>
```

Include dependency graph for `simpliedaycounter.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [SimpleDayCounter](#)
Simple day counter for reproducing theoretical calculations.

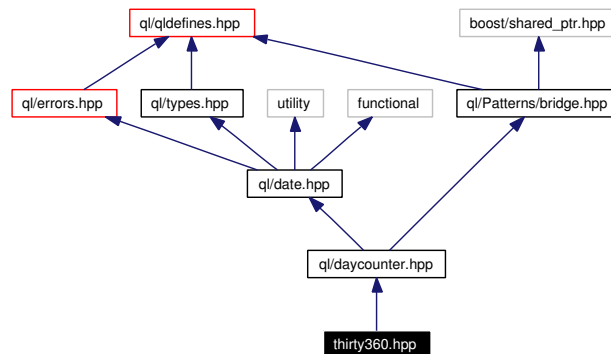
8.63 ql/DayCounters/thirty360.hpp File Reference

8.63.1 Detailed Description

30/360 day counters

```
#include <ql/daycounter.hpp>
```

Include dependency graph for thirty360.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Thirty360](#)
30/360 day count convention

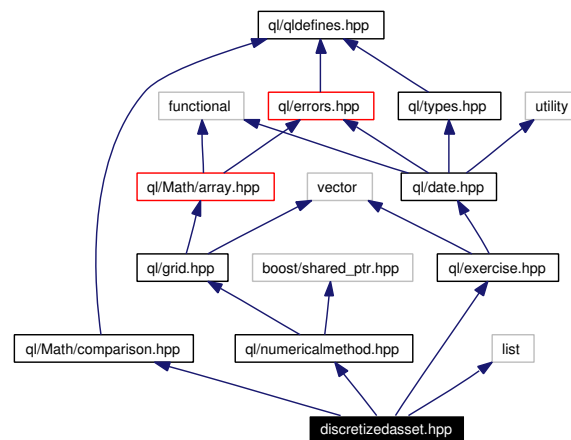
8.64 ql/discretizedasset.hpp File Reference

8.64.1 Detailed Description

Discretized asset classes.

```
#include <ql/numericalmethod.hpp>
#include <ql/Math/comparison.hpp>
#include <ql/exercise.hpp>
#include <list>
```

Include dependency graph for discretizedasset.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DiscretizedAsset](#)
Discretized asset class used by numerical methods.
- class [DiscretizedDiscountBond](#)
Useful discretized discount bond asset.
- class [DiscretizedOption](#)
Discretized option on a given asset.

8.65 ql/disposable.hpp File Reference

8.65.1 Detailed Description

generic disposable object with move semantics

```
#include <ql/qldefines.hpp>
```

Include dependency graph for disposable.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Disposable](#)
generic disposable object with move semantics

8.66 ql/errors.hpp File Reference

8.66.1 Detailed Description

Classes and functions for error handling.

```
#include <ql/qldefines.hpp>
```

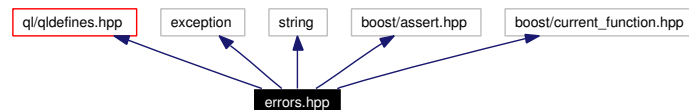
```
#include <exception>
```

```
#include <string>
```

```
#include <boost/assert.hpp>
```

```
#include <boost/current_function.hpp>
```

Include dependency graph for errors.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Error**

Base error class.

Defines

- #define **QL_FAIL**(message) throw **QuantLib::Error**(__FILE__, __LINE__, BOOST_CURRENT_FUNCTION, message)
throw an error (possibly with file and line information)
- #define **QL_ASSERT**(condition, message)
throw an error if the given condition is not verified
- #define **QL_REQUIRE**(condition, message)
throw an error if the given pre-condition is not verified
- #define **QL_ENSURE**(condition, message)
throw an error if the given post-condition is not verified

8.66.2 Define Documentation

8.66.2.1 #define QL_ASSERT(condition, message)

Value:

```
if (!(condition)) \  
    throw QuantLib::Error(__FILE__, __LINE__, BOOST_CURRENT_FUNCTION, message); \  
else
```

throw an error if the given condition is not verified

8.66.2.2 #define QL_REQUIRE(condition, message)

Value:

```
if (!(condition)) \  
    throw QuantLib::Error(__FILE__, __LINE__, BOOST_CURRENT_FUNCTION, message); \  
else
```

throw an error if the given pre-condition is not verified

Examples:

[DiscreteHedging.cpp](#), and [swapvaluation.cpp](#).

8.66.2.3 #define QL_ENSURE(condition, message)

Value:

```
if (!(condition)) \  
    throw QuantLib::Error(__FILE__, __LINE__, BOOST_CURRENT_FUNCTION, message); \  
else
```

throw an error if the given post-condition is not verified

8.67 ql/exchangerate.hpp File Reference

8.67.1 Detailed Description

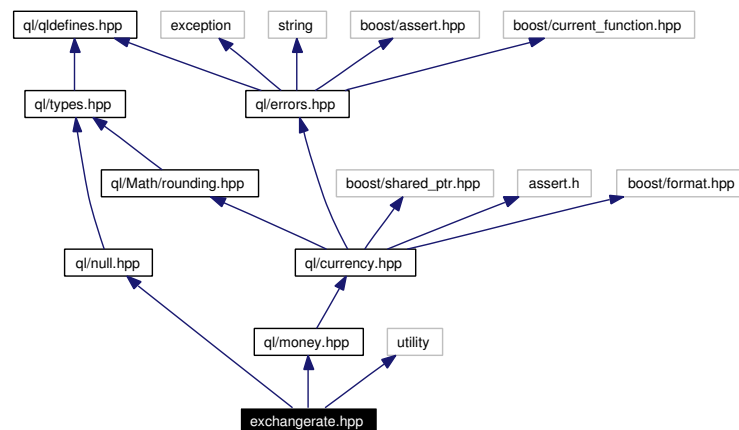
exchange rate between two currencies

```
#include <ql/money.hpp>
```

```
#include <ql/null.hpp>
```

```
#include <utility>
```

Include dependency graph for exchangerate.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ExchangeRate](#)
exchange rate between two currencies

8.68 ql/exercise.hpp File Reference

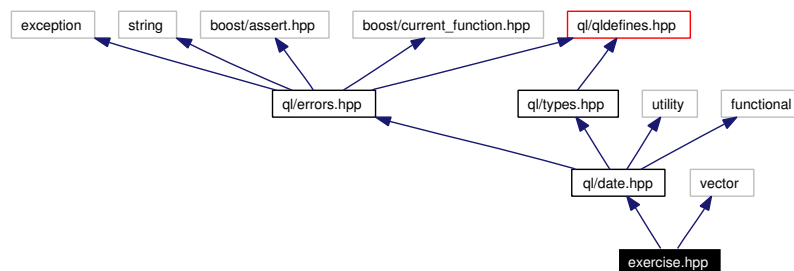
8.68.1 Detailed Description

Option exercise classes and payoff function.

```
#include <ql/date.hpp>
```

```
#include <vector>
```

Include dependency graph for exercise.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Exercise](#)
Base exercise class.
- class [EarlyExercise](#)
Early-exercise base class.
- class [AmericanExercise](#)
American exercise.
- class [BermudanExercise](#)
Bermudan exercise.
- class [EuropeanExercise](#)
European exercise.

8.69 ql/FiniteDifferences/americancondition.hpp File Reference

8.69.1 Detailed Description

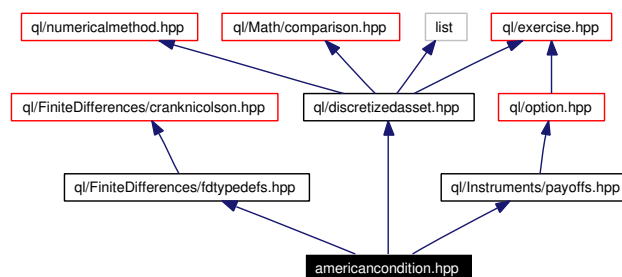
american option exercise condition

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for americancondition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AmericanCondition](#)
American exercise condition.

8.70 ql/FiniteDifferences/boundarycondition.hpp File Reference

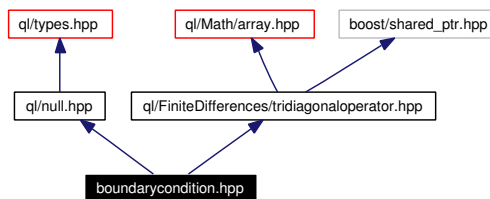
8.70.1 Detailed Description

boundary conditions for differential operators

```
#include <ql/null.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for boundarycondition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BoundaryCondition](#)
Abstract boundary condition class for finite difference problems.
- class [NeumannBC](#)
Neumann boundary condition (i.e., constant derivative).
- class [DirichletBC](#)
Neumann boundary condition (i.e., constant value).

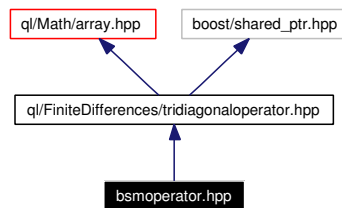
8.71 ql/FiniteDifferences/bsmoperator.hpp File Reference

8.71.1 Detailed Description

differential operator for Black-Scholes-Merton equation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for bsmoperator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BSMOperator](#)
Black-Scholes-Merton differential operator.

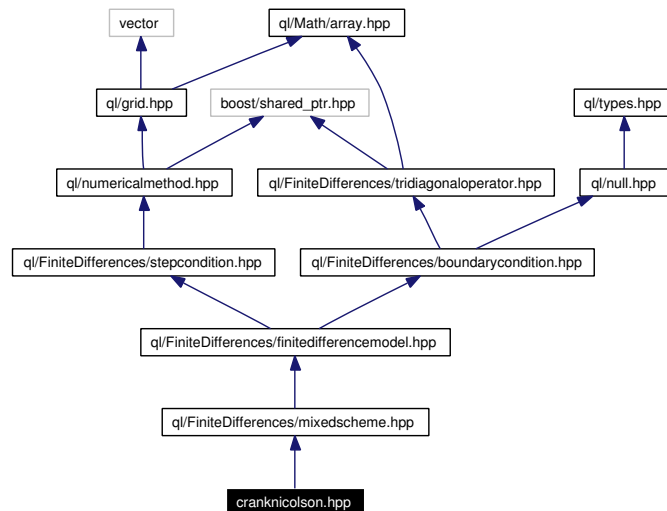
8.72 ql/FiniteDifferences/cranknicolson.hpp File Reference

8.72.1 Detailed Description

Crank-Nicolson scheme for finite difference methods.

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for cranknicolson.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CrankNicolson](#)
Crank-Nicolson scheme for finite difference methods.

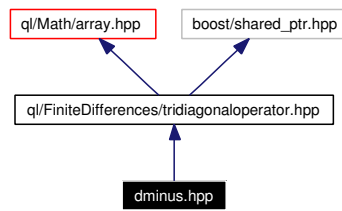
8.73 ql/FiniteDifferences/dminus.hpp File Reference

8.73.1 Detailed Description

D_ matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dminus.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DMinus](#)
D_ matricial representation

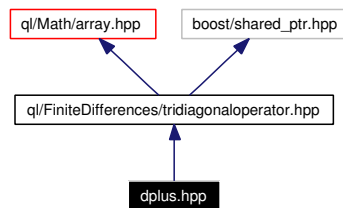
8.74 ql/FiniteDifferences/dplus.hpp File Reference

8.74.1 Detailed Description

D_+ matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dplus.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DPlus**
 D_+ matricial representation

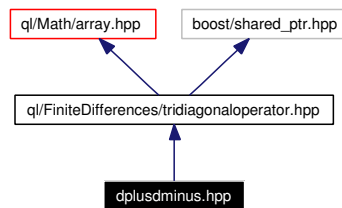
8.75 ql/FiniteDifferences/dplusdminus.hpp File Reference

8.75.1 Detailed Description

D_+D_- matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dplusdminus.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DPlusDMinus](#)
 D_+D_- matricial representation

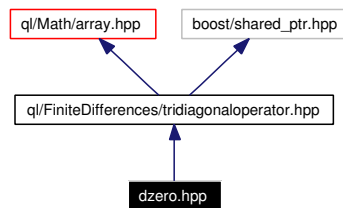
8.76 ql/FiniteDifferences/dzero.hpp File Reference

8.76.1 Detailed Description

D_0 matricial representation

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

Include dependency graph for dzero.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DZero**
 D_0 matricial representation

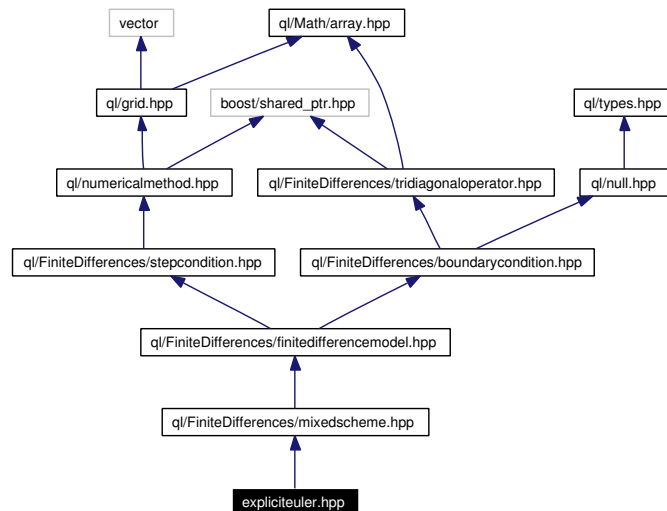
8.77 ql/FiniteDifferences/expliciteuler.hpp File Reference

8.77.1 Detailed Description

explicit Euler scheme for finite difference methods

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for expliciteuler.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ExplicitEuler](#)
Forward Euler scheme for finite difference methods.

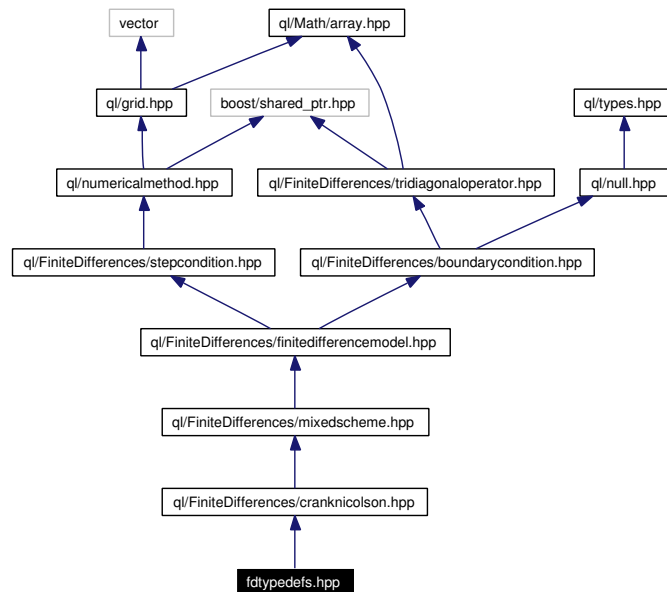
8.78 ql/FiniteDifferences/fdtypedefs.hpp File Reference

8.78.1 Detailed Description

default choices for template instantiations

```
#include <ql/FiniteDifferences/cranknicolson.hpp>
```

Include dependency graph for fdtypedefs.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef FiniteDifferenceModel< CrankNicolson< TridiagonalOperator > > [StandardFiniteDifferenceModel](#)
default choice for finite-difference model
- typedef StepCondition< Array > [StandardStepCondition](#)
default choice for step condition

8.79 ql/FiniteDifferences/finitedifferencemodel.hpp File Reference

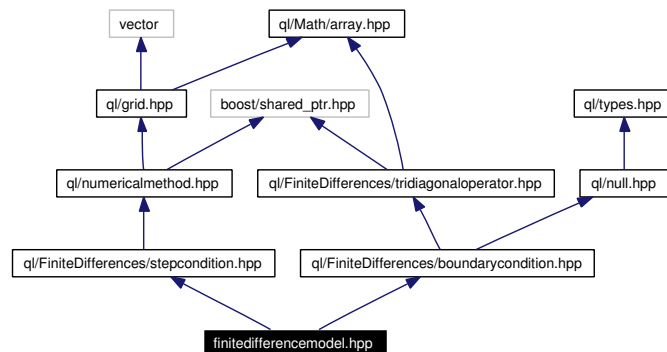
8.79.1 Detailed Description

generic finite difference model

```
#include <ql/FiniteDifferences/stepcondition.hpp>
```

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Include dependency graph for finitedifferencemodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class `FiniteDifferenceModel`
Generic finite difference model.

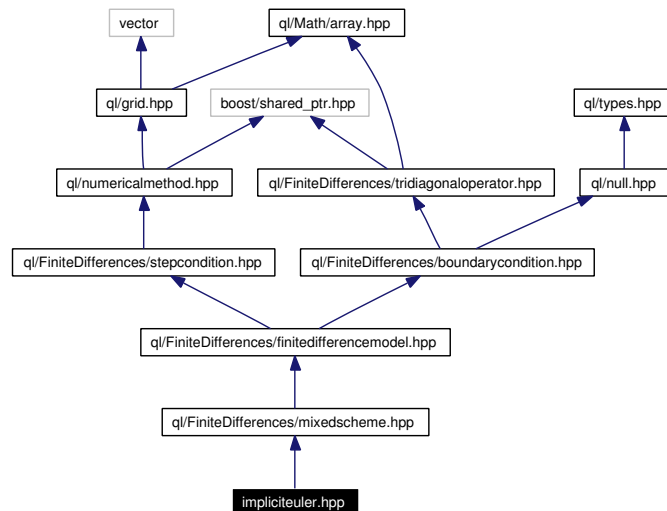
8.80 ql/FiniteDifferences/impliciteuler.hpp File Reference

8.80.1 Detailed Description

implicit Euler scheme for finite difference methods

```
#include <ql/FiniteDifferences/mixedscheme.hpp>
```

Include dependency graph for impliciteuler.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ImplicitEuler](#)
Backward Euler scheme for finite difference methods.

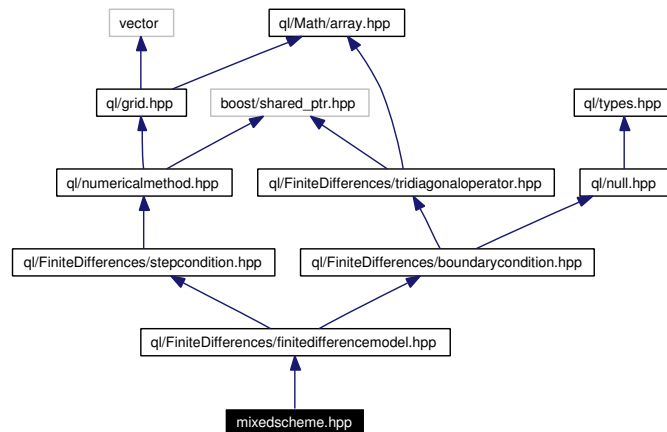
8.81 ql/FiniteDifferences/mixedscheme.hpp File Reference

8.81.1 Detailed Description

Mixed (explicit/implicit) scheme for finite difference methods.

```
#include <ql/FiniteDifferences/finitedifferencemodel.hpp>
```

Include dependency graph for mixedscheme.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MixedScheme](#)

Mixed (explicit/implicit) scheme for finite difference methods.

8.82 ql/FiniteDifferences/onefactoroperator.hpp File Reference

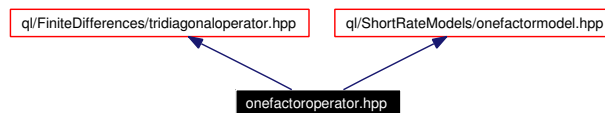
8.82.1 Detailed Description

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**

Classes

- class [OneFactorOperator](#)
Interest-rate single factor model differential operator.

8.83 ql/FiniteDifferences/shoutcondition.hpp File Reference

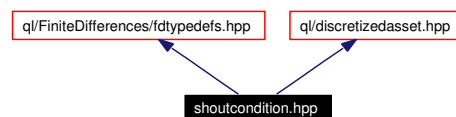
8.83.1 Detailed Description

shout option exercise condition

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for shoutcondition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ShoutCondition](#)
Shout option condition.

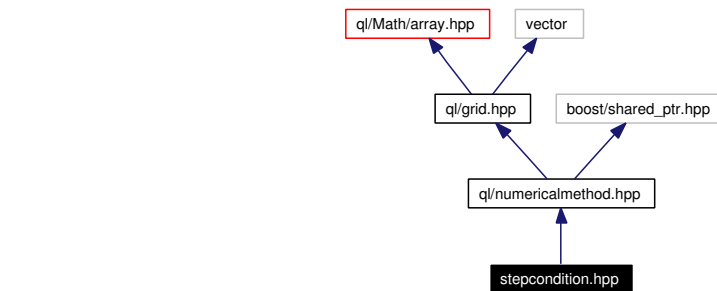
8.84 ql/FiniteDifferences/stepcondition.hpp File Reference

8.84.1 Detailed Description

conditions to be applied at every time step

```
#include <ql/numericalmethod.hpp>
```

Include dependency graph for stepcondition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [StepCondition](#)
condition to be applied at every time step

8.85 ql/FiniteDifferences/tridiagonaloperator.hpp File Reference

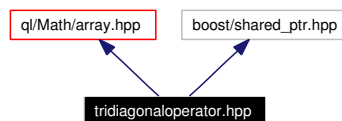
8.85.1 Detailed Description

tridiagonal operator

```
#include <ql/Math/array.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for tridiagonaloperator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TridiagonalOperator](#)
Base implementation for tridiagonal operator.
- class [TridiagonalOperator::TimeSetter](#)
encapsulation of time-setting logic

Functions

- Disposable< TridiagonalOperator > **operator+** (const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **operator-** (const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **operator+** (const TridiagonalOperator &D1, const TridiagonalOperator &D2)
- Disposable< TridiagonalOperator > **operator-** (const TridiagonalOperator &D1, const TridiagonalOperator &D2)
- Disposable< TridiagonalOperator > **operator *** ([Real](#) a, const TridiagonalOperator &D)
- Disposable< TridiagonalOperator > **operator *** (const TridiagonalOperator &D, [Real](#) a)
- Disposable< TridiagonalOperator > **operator/** (const TridiagonalOperator &D, [Real](#) a)

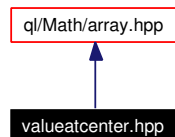
8.86 ql/FiniteDifferences/valueatcenter.hpp File Reference

8.86.1 Detailed Description

compute value, first, and second derivatives at grid center

```
#include <ql/Math/array.hpp>
```

Include dependency graph for valueatcenter.hpp:



Namespaces

- namespace **QuantLib**

Functions

- [Real valueAtCenter](#) (const Array &a)
- [Real firstDerivativeAtCenter](#) (const Array &a, const Array &grid)
- [Real secondDerivativeAtCenter](#) (const Array &a, const Array &grid)

8.86.2 Function Documentation

8.86.2.1 [Real valueAtCenter](#) (const Array &a)

mid-point value

Todo

replace with a more general (not "centered") function: `valueAt(Real spot, const Array& a);`

8.86.2.2 [Real firstDerivativeAtCenter](#) (const Array &a, const Array &grid)

mid-point first derivative

Todo

replace with a more general (not "centered") function: `firstDerivativeAt(Real spot, const Array& a, const Array& grid);`

8.86.2.3 [Real secondDerivativeAtCenter](#) (const Array &a, const Array &grid)

mid-point second derivative

Todo

replace with a more general (not "centered") function: `secondDerivativeAt(Real spot, const Array& a, const Array& grid);`

8.87 ql/grid.hpp File Reference

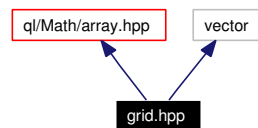
8.87.1 Detailed Description

Grid classes with useful constructors for trees and finite diffs.

```
#include <ql/Math/array.hpp>
```

```
#include <vector>
```

Include dependency graph for grid.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **TimeGrid**
time grid class

Functions

- Disposable< Array > **CenteredGrid** (**Real** center, **Real** dx, **Size** steps)
- Disposable< Array > **BoundedGrid** (**Real** xMin, **Real** xMax, **Size** steps)

8.88 ql/history.hpp File Reference

8.88.1 Detailed Description

history class

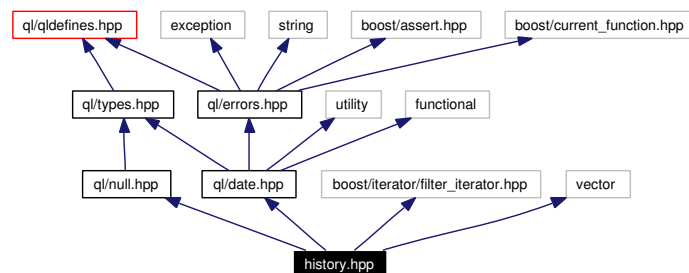
```
#include <ql/null.hpp>
```

```
#include <ql/date.hpp>
```

```
#include <boost/iterator/filter_iterator.hpp>
```

```
#include <vector>
```

Include dependency graph for history.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [History](#)
Container for historical data.
- class [History::Entry](#)
single datum in history
- class [History::const_iterator](#)
random access iterator on history entries

8.89 ql/index.hpp File Reference

8.89.1 Detailed Description

purely virtual base class for indexes

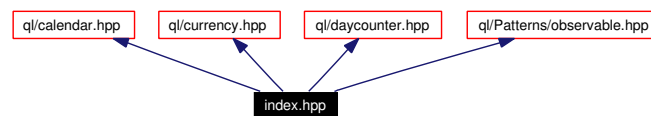
```
#include <ql/calendar.hpp>
```

```
#include <ql/currency.hpp>
```

```
#include <ql/daycounter.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for index.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Index](#)
purely virtual base class for indexes

8.90 ql/Indexes/audlibor.hpp File Reference

8.90.1 Detailed Description

AUD Libor index (check settlement days)

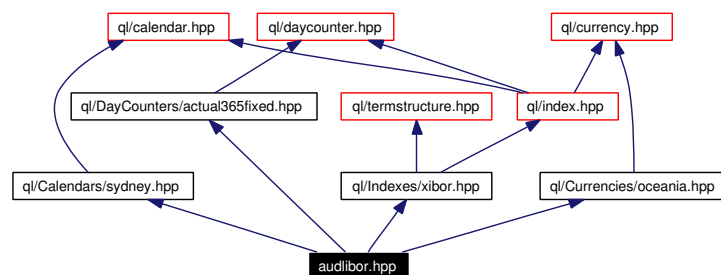
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/sydney.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

```
#include <ql/Currencies/oceania.hpp>
```

Include dependency graph for audlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AUDLibor](#)
AUD Libor index, also known as SIBOR

8.91 ql/Indexes/cadlibor.hpp File Reference

8.91.1 Detailed Description

CAD Libor index (Also known as CDOR)

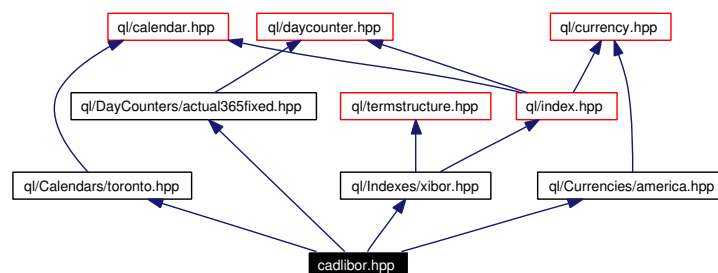
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/toronto.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

```
#include <ql/Currencies/america.hpp>
```

Include dependency graph for cadlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CADLibor**
CAD Libor index, also known as CDOR

8.92 ql/Indexes/chflibor.hpp File Reference

8.92.1 Detailed Description

CHF Libor index (Also known as ZIBOR)

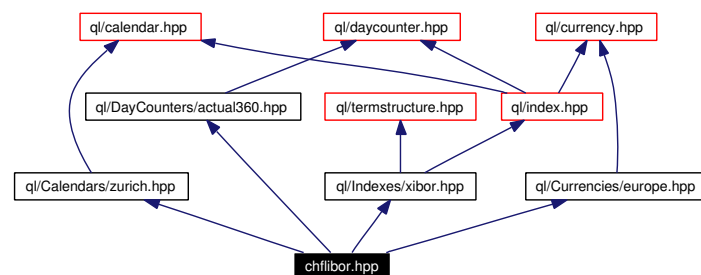
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/zurich.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for chflibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **CHFLibor**
CHF Libor index, also known as ZIBOR

8.93 ql/Indexes/euribor.hpp File Reference

8.93.1 Detailed Description

Euribor index

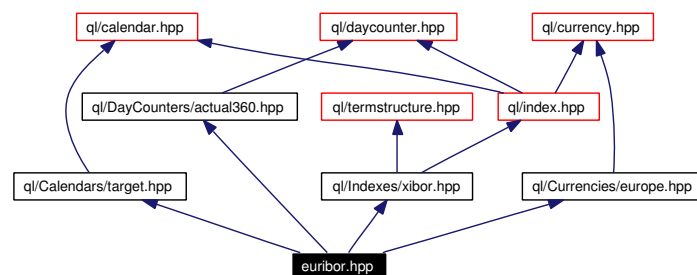
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/target.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

```
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for euribor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Euribor**
Euribor index

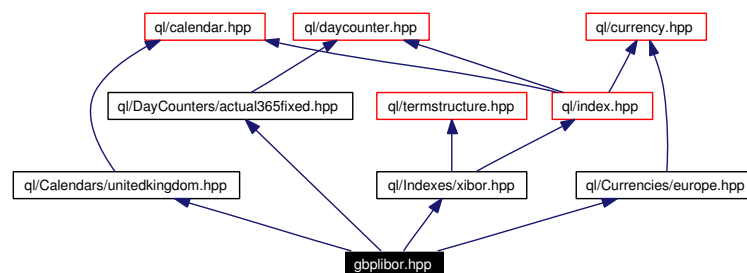
8.94 ql/Indexes/gbplibor.hpp File Reference

8.94.1 Detailed Description

GBP Libor index

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/unitedkingdom.hpp>
#include <ql/DayCounters/actual365fixed.hpp>
#include <ql/Currencies/europe.hpp>
```

Include dependency graph for gbplibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **GBPLibor**
GBP Libor index

8.95 ql/Indexes/indexmanager.hpp File Reference

8.95.1 Detailed Description

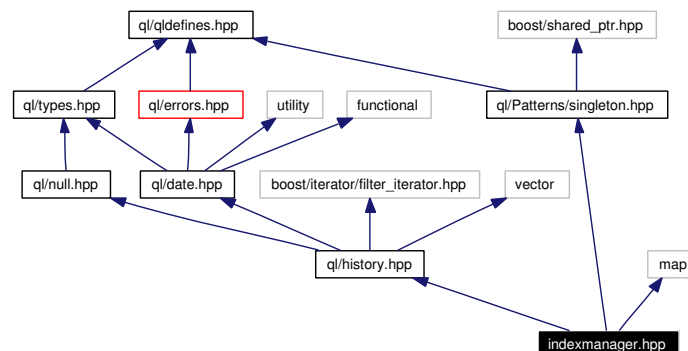
global repository for past index fixings

```
#include <ql/history.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <map>
```

Include dependency graph for indexmanager.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class `IndexManager`
global repository for past index fixings

8.96 ql/Indexes/jpylibor.hpp File Reference

8.96.1 Detailed Description

JPY Libor index (Also known as TIBOR, check settlement days)

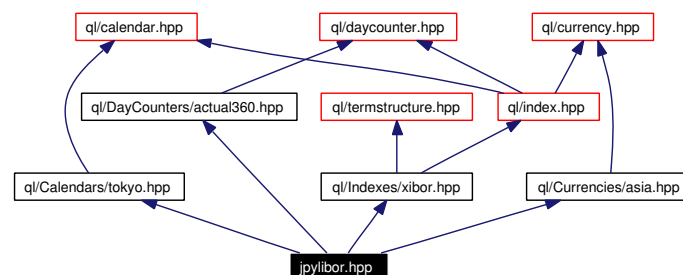
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/tokyo.hpp>
```

```
#include <ql/DayCounters/actual360.hpp>
```

```
#include <ql/Currencies/asia.hpp>
```

Include dependency graph for jpylibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **JPYLibor**
JPY Libor index, also known as TIBOR

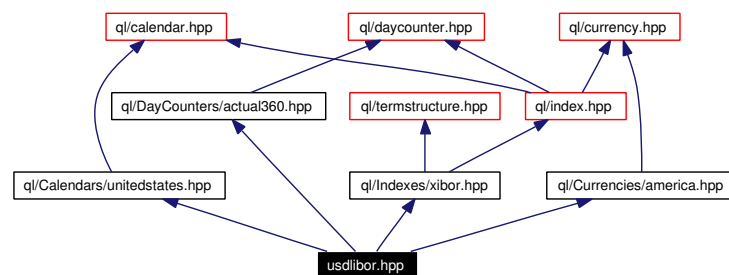
8.97 ql/Indexes/usdlibor.hpp File Reference

8.97.1 Detailed Description

USD Libor index

```
#include <ql/Indexes/xibor.hpp>
#include <ql/Calendars/unitedstates.hpp>
#include <ql/DayCounters/actual360.hpp>
#include <ql/Currencies/america.hpp>
```

Include dependency graph for usdlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **USDLibor**
USD Libor index

8.98 ql/Indexes/xibor.hpp File Reference

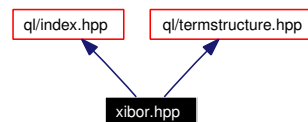
8.98.1 Detailed Description

base class for libor indexes

```
#include <ql/index.hpp>
```

```
#include <ql/termstructure.hpp>
```

Include dependency graph for xibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Xibor](#)
base class for libor indexes

8.99 ql/Indexes/xibormanager.hpp File Reference

8.99.1 Detailed Description

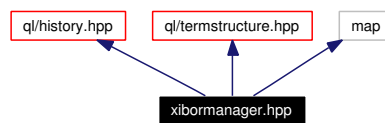
global repository for Xibor histories

```
#include <ql/history.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <map>
```

Include dependency graph for xibormanager.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [XiborManager](#)
global repository for libor histories

8.100 ql/Indexes/zarlibor.hpp File Reference

8.100.1 Detailed Description

ZAR Libor index (also known as JIBAR)

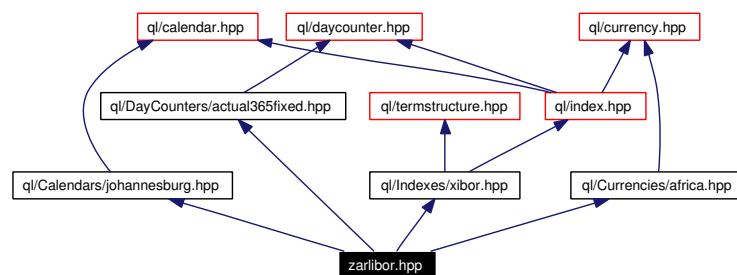
```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/Calendars/johannesburg.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

```
#include <ql/Currencies/africa.hpp>
```

Include dependency graph for zarlibor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **ZARLibor**
ZAR Libor index, also known as JIBAR

8.101 ql/instrument.hpp File Reference

8.101.1 Detailed Description

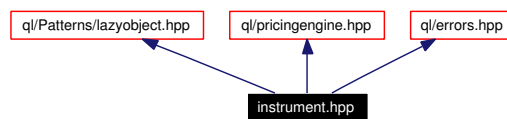
Abstract instrument class.

```
#include <ql/Patterns/lazyobject.hpp>
```

```
#include <ql/pricingengine.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for instrument.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Instrument](#)
Abstract instrument class.
- class [Value](#)
pricing results

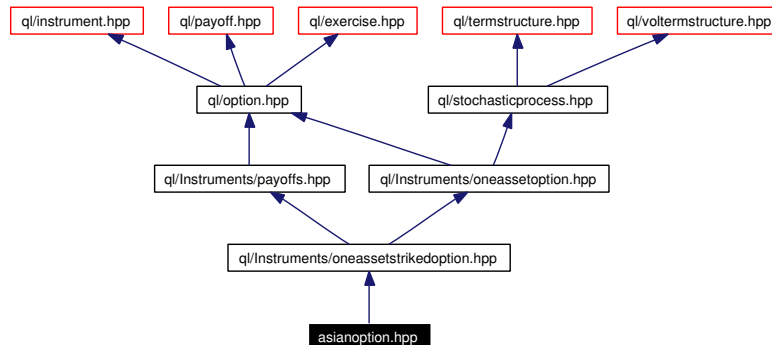
8.102 ql/Instruments/asianoption.hpp File Reference

8.102.1 Detailed Description

Asian option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for asianoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct **Average**
placeholder for enumerated averaging types
- class **ContinuousAveragingAsianOption**
Continuous-averaging Asian option.
- class **DiscreteAveragingAsianOption**
Discrete-averaging Asian option.
- class **DiscreteAveragingAsianOption::arguments**
Extra arguments for single-asset discrete-average Asian option.
- class **ContinuousAveragingAsianOption::arguments**
Extra arguments for single-asset continuous-average Asian option.
- class **DiscreteAveragingAsianOption::engine**
Discrete-averaging Asian engine base class.
- class **ContinuousAveragingAsianOption::engine**
Continuous-averaging Asian engine base class.

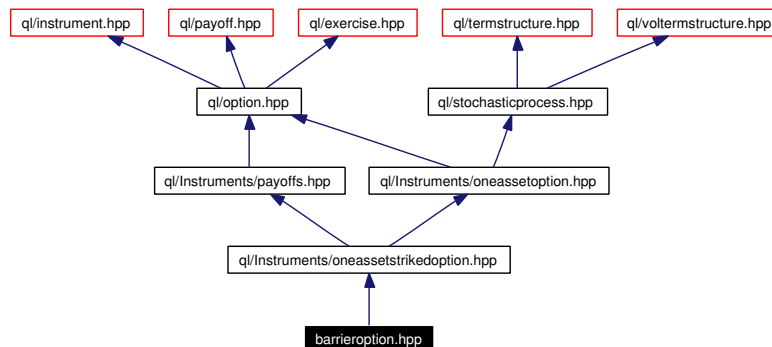
8.103 ql/Instruments/barrieroption.hpp File Reference

8.103.1 Detailed Description

Barrier option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for barrieroption.hpp:



Namespaces

- namespace `QuantLib`

Classes

- struct `Barrier`
Placeholder for enumerated barrier types.
- class `BarrierOption`
Barrier option on a single asset.
- class `BarrierOption::arguments`
Arguments for barrier option calculation
- class `BarrierOption::engine`
Barrier engine base class

8.104 ql/Instruments/basketoption.hpp File Reference

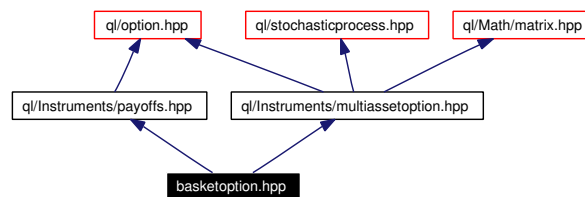
8.104.1 Detailed Description

Basket option on a number of assets.

```
#include <ql/Instruments/payoffs.hpp>
```

```
#include <ql/Instruments/multiassetoption.hpp>
```

Include dependency graph for basketoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BasketOption](#)
Basket option on a number of assets.
- class [BasketOption::arguments](#)
Arguments for basket option calculation
- class [BasketOption::engine](#)
Basket option engine base class

8.105 ql/Instruments/bond.hpp File Reference

8.105.1 Detailed Description

concrete bond class

```
#include <ql/instrument.hpp>
```

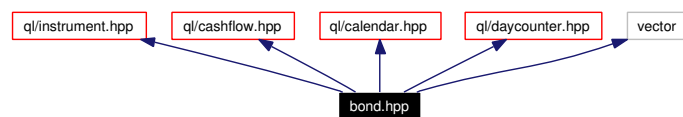
```
#include <ql/cashflow.hpp>
```

```
#include <ql/calendar.hpp>
```

```
#include <ql/daycounter.hpp>
```

```
#include <vector>
```

Include dependency graph for bond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Bond**

Base bond class.

8.106 ql/Instruments/capfloor.hpp File Reference

8.106.1 Detailed Description

Cap and Floor class.

```
#include <ql/numericalmethod.hpp>
```

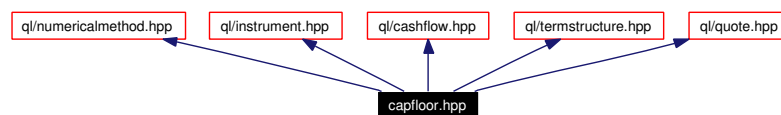
```
#include <ql/instrument.hpp>
```

```
#include <ql/cashflow.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <ql/quote.hpp>
```

Include dependency graph for capfloor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CapFloor](#)
Base class for cap-like instruments.
- class [Cap](#)
Concrete cap class.
- class [Floor](#)
Concrete floor class.
- class [Collar](#)
Concrete collar class.
- class [CapFloor::arguments](#)
Arguments for cap/floor calculation
- class [CapFloor::results](#)
Results from cap/floor calculation

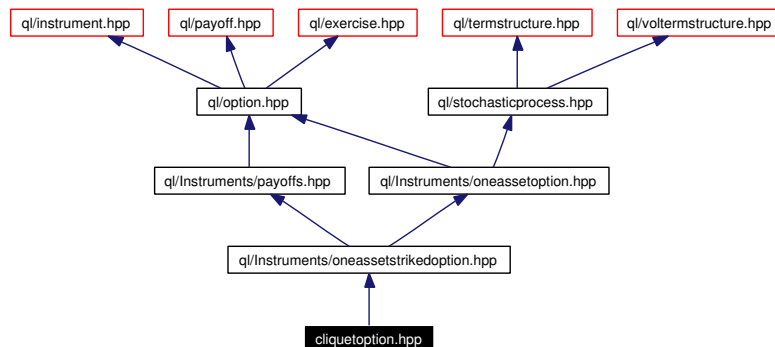
8.107 ql/Instruments/cliqetoption.hpp File Reference

8.107.1 Detailed Description

Cliquet option.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for cliqetoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CliquetOption](#)
cliquet (Ratchet) option
- class [CliquetOption::arguments](#)
Arguments for cliquet option calculation
- class [CliquetOption::engine](#)
Cliquet engine base class.

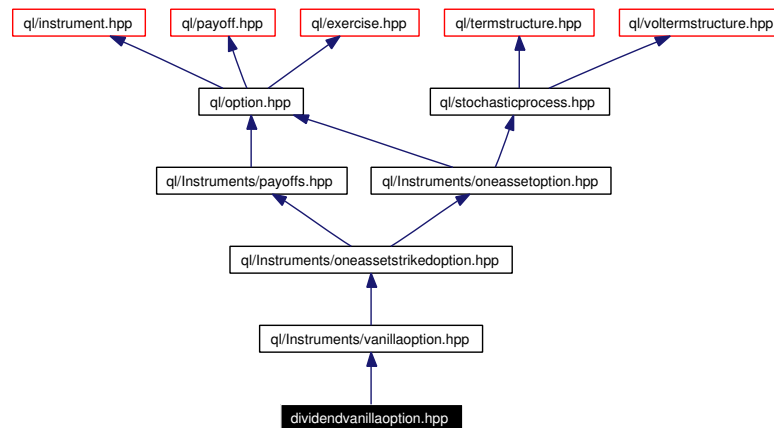
8.108 ql/Instruments/dividendvanillaoption.hpp File Reference

8.108.1 Detailed Description

Vanilla option on a single asset with discrete dividends.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for dividendvanillaoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DividendVanillaOption](#)
Single-asset vanilla option (no barriers) with discrete dividends.
- class [DividendVanillaOption::arguments](#)
Arguments for dividend vanilla option calculation
- class [DividendVanillaOption::engine](#)
Dividend vanilla option engine base class.

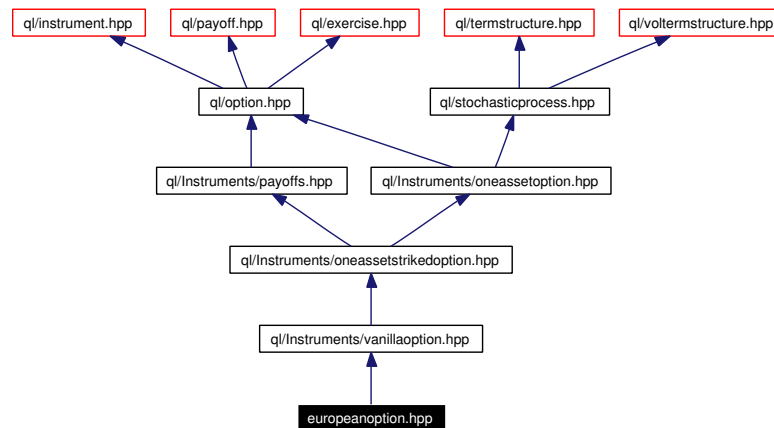
8.109 ql/Instruments/europeanoption.hpp File Reference

8.109.1 Detailed Description

European option on a single asset.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for europeanoption.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `EuropeanOption`
European option on a single asset.

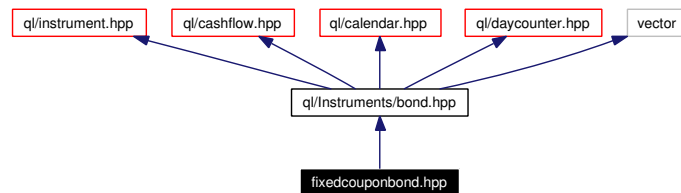
8.110 ql/Instruments/fixedcouponbond.hpp File Reference

8.110.1 Detailed Description

fixed-coupon bond

```
#include <ql/Instruments/bond.hpp>
```

Include dependency graph for fixedcouponbond.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FixedCouponBond**
fixed-coupon bond

8.111 ql/Instruments/forwardvanillaoption.hpp File Reference

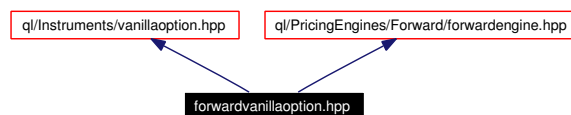
8.111.1 Detailed Description

Forward version of a vanilla option.

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

Include dependency graph for forwardvanillaoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ForwardVanillaOption](#)
Forward version of a vanilla option.

8.112 ql/Instruments/multiassetoption.hpp File Reference

8.112.1 Detailed Description

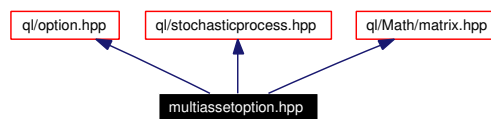
Option on multiple assets.

```
#include <ql/option.hpp>
```

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for multiassetoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MultiAssetOption](#)
Base class for options on multiple assets.
- class [MultiAssetOption::arguments](#)
Arguments for multi-asset option calculation
- class [MultiAssetOption::results](#)
Results from multi-asset option calculation

8.113 ql/Instruments/oneassetoption.hpp File Reference

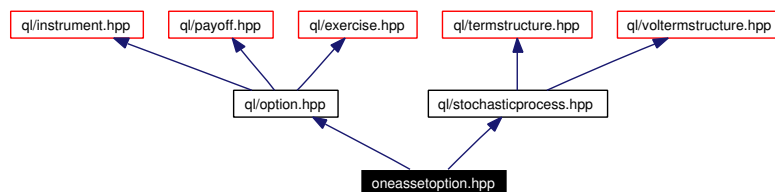
8.113.1 Detailed Description

Option on a single asset.

```
#include <ql/option.hpp>
```

```
#include <ql/stochasticprocess.hpp>
```

Include dependency graph for oneassetoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OneAssetOption](#)
Base class for options on a single asset.
- class [OneAssetOption::arguments](#)
Arguments for single-asset option calculation
- class [OneAssetOption::results](#)
Results from single-asset option calculation

8.114 ql/Instruments/oneassetstrikedoption.hpp File Reference

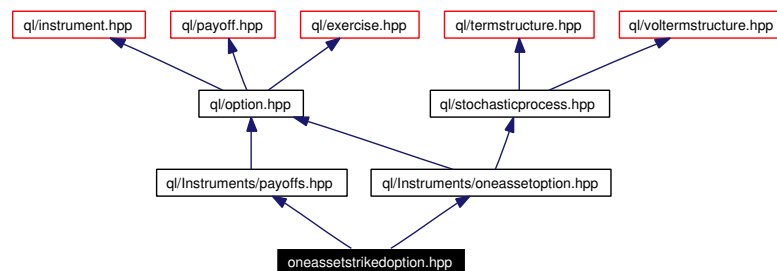
8.114.1 Detailed Description

Option on a single asset with striked payoff.

```
#include <ql/Instruments/oneassetoption.hpp>
```

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for oneassetstrikedoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OneAssetStrikedOption](#)

Base class for options on a single asset with striked payoff.

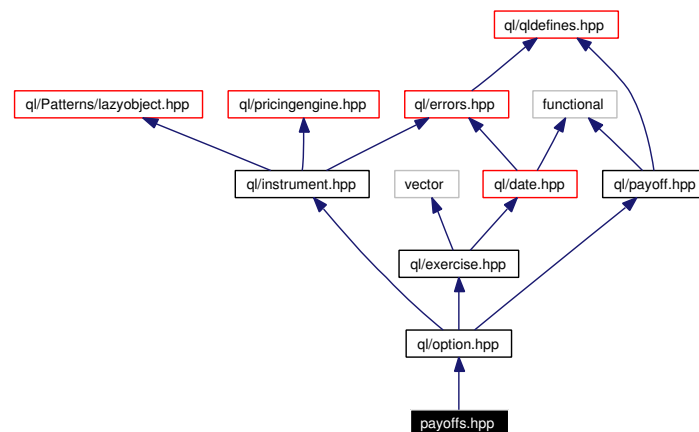
8.115 ql/Instruments/payoffs.hpp File Reference

8.115.1 Detailed Description

Payoffs for various options.

```
#include <ql/option.hpp>
```

Include dependency graph for `payoffs.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [TypePayoff](#)
Intermediate class for call/put/straddle payoffs.
- class [StrikedTypePayoff](#)
Intermediate class for payoffs based on a fixed strike.
- class [PlainVanillaPayoff](#)
Plain-vanilla payoff.
- class [PercentageStrikePayoff](#)
Payoff with strike expressed as percentage
- class [CashOrNothingPayoff](#)
Binary cash-or-nothing payoff.
- class [AssetOrNothingPayoff](#)
Binary asset-or-nothing payoff.
- class [GapPayoff](#)

Binary gap payoff.

- class [SuperSharePayoff](#)
Binary supershare payoff.

8.116 ql/Instruments/quantoforwardvanillaoption.hpp File Reference

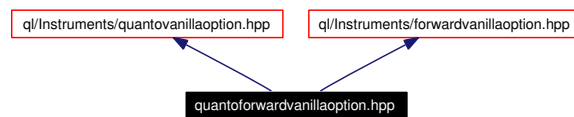
8.116.1 Detailed Description

Quanto version of a forward vanilla option.

```
#include <ql/Instruments/quantovanillaoption.hpp>
```

```
#include <ql/Instruments/forwardvanillaoption.hpp>
```

Include dependency graph for quantoforwardvanillaoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [QuantoForwardVanillaOption](#)
Quanto version of a forward vanilla option.

8.117 ql/Instruments/quantovanillaoption.hpp File Reference

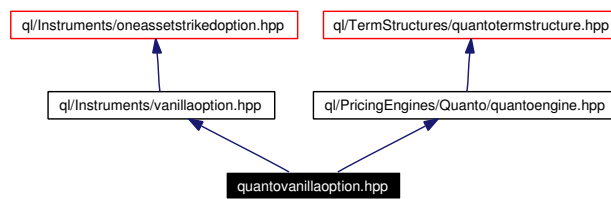
8.117.1 Detailed Description

Quanto version of a vanilla option.

```
#include <ql/Instruments/vanillaoption.hpp>
```

```
#include <ql/PricingEngines/Quanto/quantoengine.hpp>
```

Include dependency graph for quantovanillaoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [QuantoVanillaOption](#)
quanto version of a vanilla option

8.118 ql/Instruments/simpleswap.hpp File Reference

8.118.1 Detailed Description

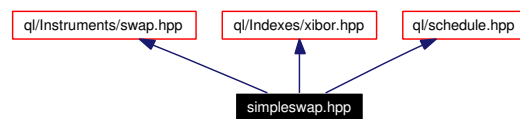
Simple fixed-rate vs Libor swap.

```
#include <ql/Instruments/swap.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

```
#include <ql/schedule.hpp>
```

Include dependency graph for simpleswap.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SimpleSwap](#)
Simple fixed-rate vs Libor swap.
- class [SimpleSwap::arguments](#)
Arguments for simple swap calculation
- class [SimpleSwap::results](#)
Results from simple swap calculation

8.119 ql/Instruments/stock.hpp File Reference

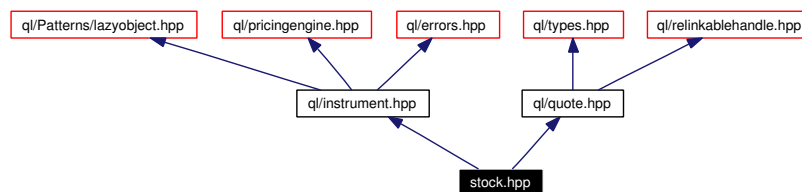
8.119.1 Detailed Description

concrete stock class

```
#include <ql/instrument.hpp>
```

```
#include <ql/quote.hpp>
```

Include dependency graph for stock.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Stock](#)
Simple stock class.

8.120 ql/Instruments/swap.hpp File Reference

8.120.1 Detailed Description

Interest rate swap.

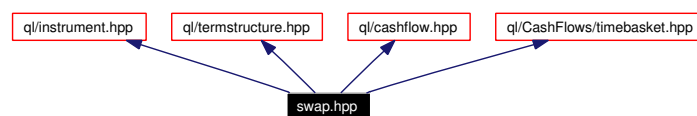
```
#include <ql/instrument.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <ql/cashflow.hpp>
```

```
#include <ql/CashFlows/timebasket.hpp>
```

Include dependency graph for swap.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Swap](#)
Interest rate swap.

8.121 ql/Instruments/swaption.hpp File Reference

8.121.1 Detailed Description

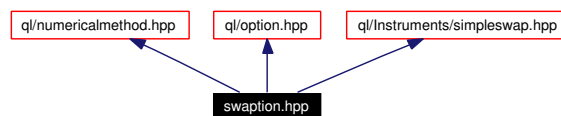
Swaption class.

```
#include <ql/numericalmethod.hpp>
```

```
#include <ql/option.hpp>
```

```
#include <ql/Instruments/simpleswap.hpp>
```

Include dependency graph for swaption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Swaption](#)
Swaption class
- class [Swaption::arguments](#)
Arguments for swaption calculation
- class [Swaption::results](#)
Results from swaption calculation

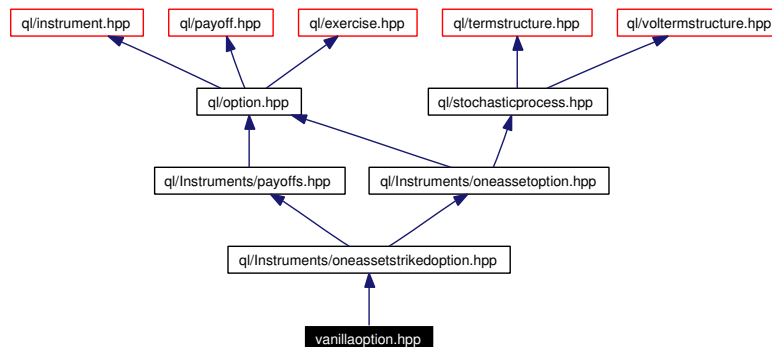
8.122 ql/Instruments/vanillaoption.hpp File Reference

8.122.1 Detailed Description

Vanilla option on a single asset.

```
#include <ql/Instruments/oneassetstrikedoption.hpp>
```

Include dependency graph for vanillaoption.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `VanillaOption`
Vanilla option (no discrete dividends, no barriers) on a single asset.
- class `VanillaOption::engine`
Vanilla option engine base class.

8.123 ql/interestrate.hpp File Reference

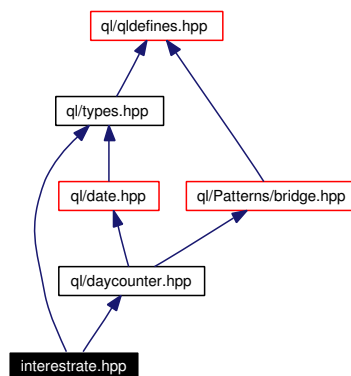
8.123.1 Detailed Description

Instrument rate class.

```
#include <ql/types.hpp>
```

```
#include <ql/daycounter.hpp>
```

Include dependency graph for interestrate.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InterestRate](#)
Concrete interest rate class.
- class [CompoundingRuleFormatter](#)
Formats compounding rule for output.
- class [InterestRateFormatter](#)
Formats interest rates for output.

Enumerations

- enum **Compounding** { [Simple](#) = 0, [Compounded](#) = 1, [Continuous](#) = 2, [SimpleThenCompounded](#) }
Interest rate compounding rule.

8.124 ql/Lattices/binomialtree.hpp File Reference

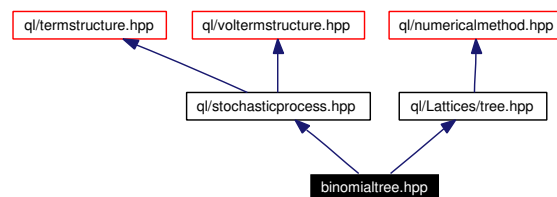
8.124.1 Detailed Description

Binomial tree class.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/Lattices/tree.hpp>
```

Include dependency graph for binomialtree.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BinomialTree](#)
Binomial tree base class.
- class [EqualProbabilitiesBinomialTree](#)
Base class for equal probabilities binomial tree.
- class [EqualJumpsBinomialTree](#)
Base class for equal jumps binomial tree.
- class [JarrowRudd](#)
Jarrow-Rudd (multiplicative) equal probabilities binomial tree.
- class [CoxRossRubinstein](#)
Cox-Ross-Rubinstein (multiplicative) equal jumps binomial tree.
- class [AdditiveEQPBinomialTree](#)
Additive equal probabilities binomial tree.
- class [Trigeorgis](#)
Trigeorgis (additive equal jumps) binomial tree
- class [Tian](#)
Tian tree: third moment matching, multiplicative approach
- class [LeisenReimer](#)

Leisen & Reimer tree: multiplicative approach.

8.125 ql/Lattices/bsmlattice.hpp File Reference

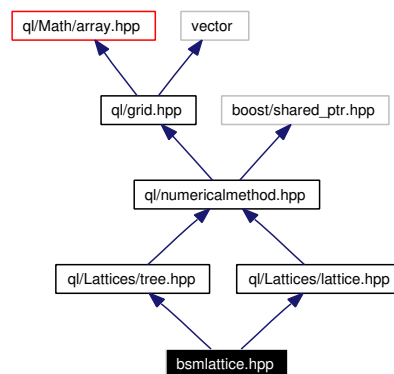
8.125.1 Detailed Description

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**

Classes

- class [BlackScholesLattice](#)

Simple binomial lattice approximating the Black-Scholes model.

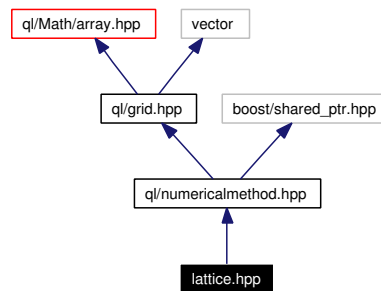
8.126 ql/Lattices/lattice.hpp File Reference

8.126.1 Detailed Description

Lattice method class.

```
#include <ql/numericalmethod.hpp>
```

Include dependency graph for lattice.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Lattice](#)
Lattice-method base class.

8.127 ql/Lattices/lattice2d.hpp File Reference

8.127.1 Detailed Description

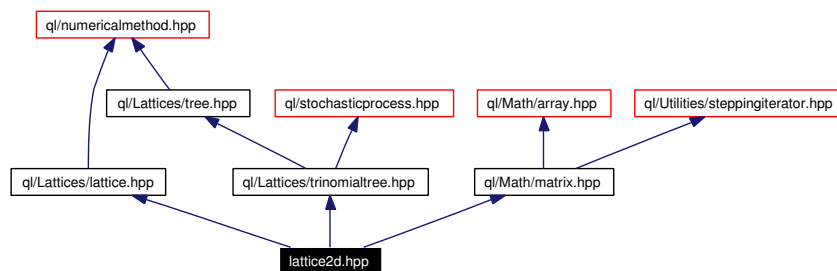
Two-dimensional lattice class.

```
#include <ql/Lattices/lattice.hpp>
```

```
#include <ql/Lattices/trinomialtree.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for lattice2d.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Lattice2D](#)

Two-dimensional lattice.

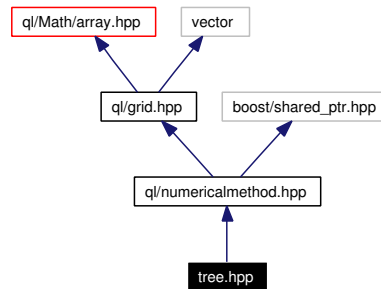
8.128 ql/Lattices/tree.hpp File Reference

8.128.1 Detailed Description

Tree class.

```
#include <ql/numericalmethod.hpp>
```

Include dependency graph for tree.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Tree](#)
Tree approximating a single-factor diffusion

8.129 ql/Lattices/trinomialtree.hpp File Reference

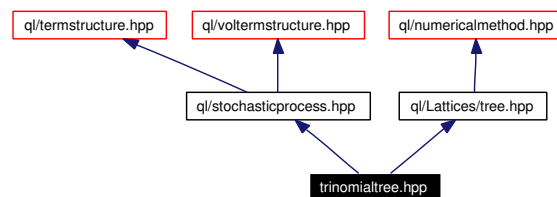
8.129.1 Detailed Description

Trinomial tree class.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/Lattices/tree.hpp>
```

Include dependency graph for trinomialtree.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TrinomialTree](#)
Recombining trinomial tree class.
- class [TrinomialBranching](#)
Branching scheme for a trinomial node.

8.130 ql/Math/array.hpp File Reference

8.130.1 Detailed Description

1-D array used in linear algebra.

```
#include <ql/errors.hpp>
```

```
#include <ql/disposable.hpp>
```

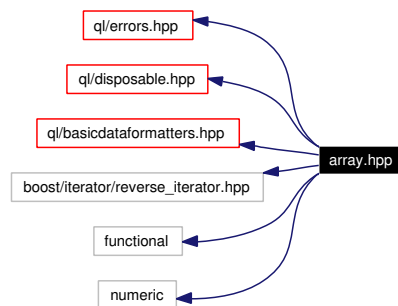
```
#include <ql/basicdataformatters.hpp>
```

```
#include <boost/iterator/reverse_iterator.hpp>
```

```
#include <functional>
```

```
#include <numeric>
```

Include dependency graph for array.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Array](#)
1-D array used in linear algebra.
- class [ArrayFormatter](#)
format arrays for output

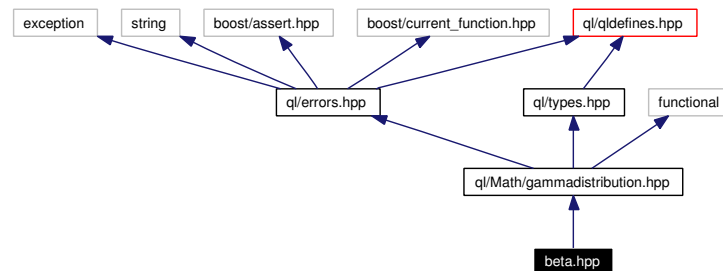
8.131 ql/Math/beta.hpp File Reference

8.131.1 Detailed Description

Beta and beta incomplete functions.

```
#include <ql/Math/gammadistribution.hpp>
```

Include dependency graph for beta.hpp:



Namespaces

- namespace **QuantLib**

Functions

- **Real** `betaFunction` (**Real** `z`, **Real** `w`)
- **Real** `betaContinuedFraction` (**Real** `a`, **Real** `b`, **Real** `x`, **Real** `accuracy=1e-16`, **Integer** `maxIteration=100`)
- **Real** `incompleteBetaFunction` (**Real** `a`, **Real** `b`, **Real** `x`, **Real** `accuracy=1e-16`, **Integer** `maxIteration=100`)

Incomplete Beta function.

8.131.2 Function Documentation

8.131.2.1 **Real** `incompleteBetaFunction` (**Real** `a`, **Real** `b`, **Real** `x`, **Real** `accuracy = 1e-16`, **Integer** `maxIteration = 100`)

Incomplete Beta function.

Incomplete Beta function

The implementation of the algorithm was inspired by "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery, chapter 6

8.132 ql/Math/bicubicsplineinterpolation.hpp File Reference

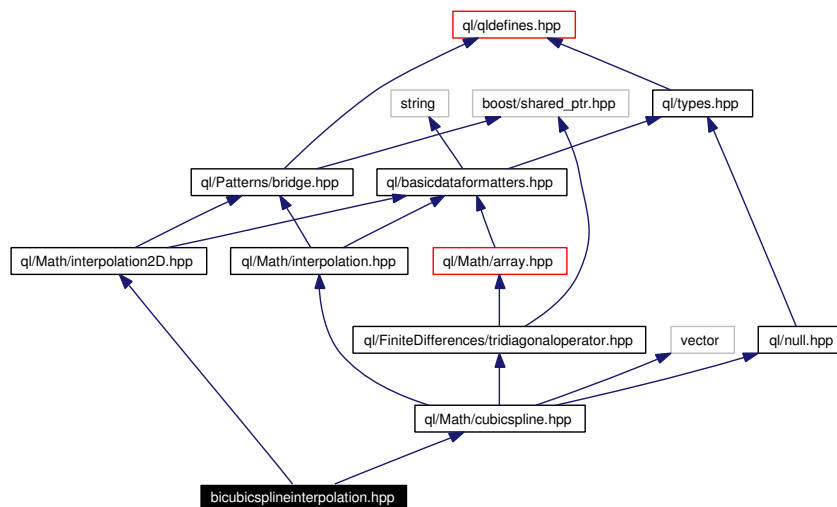
8.132.1 Detailed Description

bicubic spline interpolation between discrete points

```
#include <ql/Math/interpolation2D.hpp>
```

```
#include <ql/Math/cubicspline.hpp>
```

Include dependency graph for bicubicsplineinterpolation.hpp:



Namespaces

- namespace `QuantLib`
- namespace `QuantLib::detail`

Classes

- class `BicubicSpline`

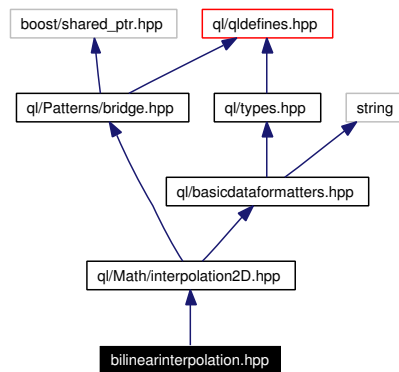
8.133 ql/Math/bilinearinterpolation.hpp File Reference

8.133.1 Detailed Description

bilinear interpolation between discrete points

```
#include <ql/Math/interpolation2D.hpp>
```

Include dependency graph for bilinearinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [BilinearInterpolation](#)
bilinear interpolation between discrete points

8.134 ql/Math/binomialdistribution.hpp File Reference

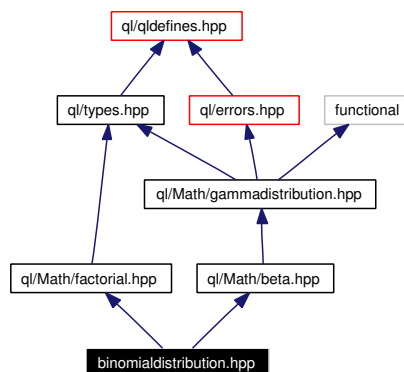
8.134.1 Detailed Description

Binomial distribution.

```
#include <ql/Math/factorial.hpp>
```

```
#include <ql/Math/beta.hpp>
```

Include dependency graph for binomialdistribution.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BinomialDistribution](#)
Binomial probability distribution function.
- class [CumulativeBinomialDistribution](#)
Cumulative binomial distribution function.

Functions

- [Real binomialCoefficientLn](#) (BigNatural n, BigNatural k)
- [Real binomialCoefficient](#) (BigNatural n, BigNatural k)
- [Real PeizerPrattMethod2Inversion](#) (Real z, BigNatural n)

8.134.2 Function Documentation

8.134.2.1 [Real PeizerPrattMethod2Inversion](#) (Real z, BigNatural n)

Given an odd integer n and a real number z it returns p such that: $1 - \text{CumulativeBinomialDistribution}((n-1)/2, n, p) = \text{CumulativeNormalDistribution}(z)$

Precondition:

n must be odd

8.135 ql/Math/bivariatenormaldistribution.hpp File Reference

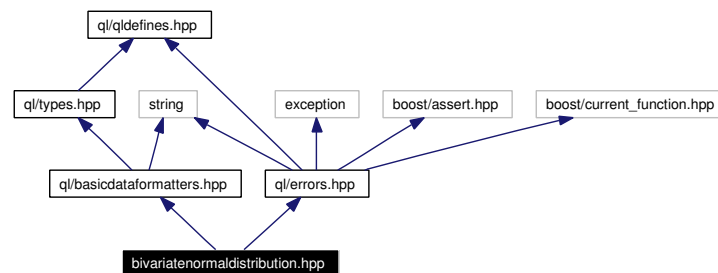
8.135.1 Detailed Description

bivariate cumulative normal distribution

```
#include <ql/basicdataformatters.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for bivariatenormaldistribution.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BivariateCumulativeNormalDistribution](#)
Cumulative bivariate normal distribution function.

8.136 ql/Math/chisquaredistribution.hpp File Reference

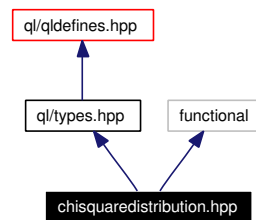
8.136.1 Detailed Description

Chi-square (central and non-central) distributions.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for chisquaredistribution.hpp:



Namespaces

- namespace **QuantLib**

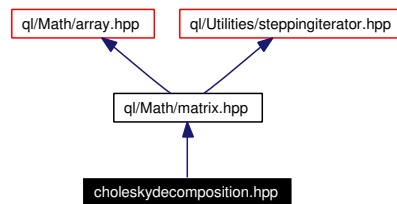
8.137 ql/Math/choleskydecomposition.hpp File Reference

8.137.1 Detailed Description

Cholesky decomposition.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for choleskydecomposition.hpp:



Namespaces

- namespace **QuantLib**

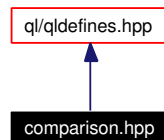
8.138 ql/Math/comparison.hpp File Reference

8.138.1 Detailed Description

floating-point comparisons

```
#include <ql/qldefines.hpp>
```

Include dependency graph for comparison.hpp:



Namespaces

- namespace QuantLib

Functions

- bool `close` (Real `x`, Real `y`, Size `n=42`)
- bool `close_enough` (Real `x`, Real `y`, Size `n=42`)

8.138.2 Function Documentation

8.138.2.1 bool `close` (Real `x`, Real `y`, Size `n = 42`)

Follows somewhat the advice of Knuth on checking for floating-point equality. The closeness relationship is:

$$\text{close}(x, y, n) \equiv |x - y| \leq \varepsilon|x| \wedge |x - y| \leq \varepsilon|y|$$

where ε is n times the machine accuracy.

8.138.2.2 bool `close_enough` (Real `x`, Real `y`, Size `n = 42`)

Follows somewhat the advice of Knuth on checking for floating-point equality. The closeness relationship is:

$$\text{close}(x, y, n) \equiv |x - y| \leq \varepsilon|x| \vee |x - y| \leq \varepsilon|y|$$

where ε is n times the machine accuracy.

8.139 ql/Math/cubicspline.hpp File Reference

8.139.1 Detailed Description

cubic spline interpolation between discrete points

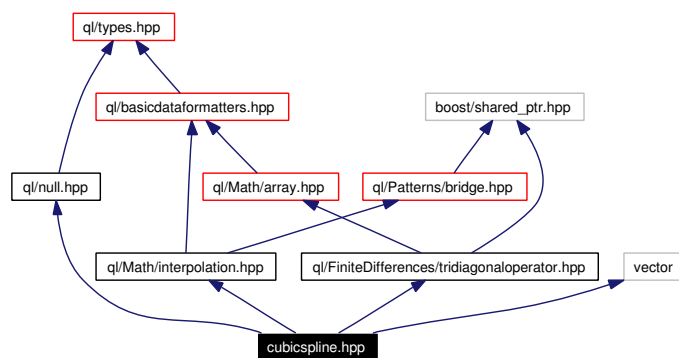
```
#include <ql/Math/interpolation.hpp>
```

```
#include <ql/FiniteDifferences/tridiagonaloperator.hpp>
```

```
#include <ql/null.hpp>
```

```
#include <vector>
```

Include dependency graph for cubicspline.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [CubicSpline](#)
Cubic spline interpolation between discrete points.
- class [MonotonicCubicSpline](#)
Cubic spline with monotonicity constraint
- class [NaturalCubicSpline](#)
Cubic spline with null second derivative at end points
- class [NaturalMonotonicCubicSpline](#)
Natural cubic spline with monotonicity constraint.

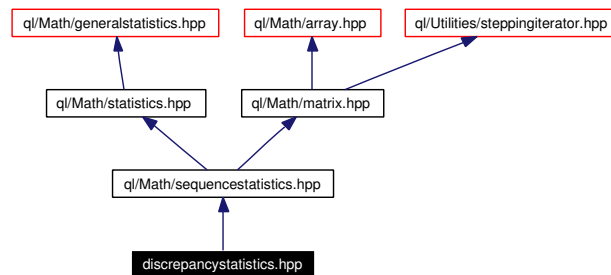
8.140 ql/Math/discrepancystatistics.hpp File Reference

8.140.1 Detailed Description

Statistic tool for sequences with discrepancy calculation.

```
#include <ql/Math/sequencestatistics.hpp>
```

Include dependency graph for `discrepancystatistics.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [DiscrepancyStatistics](#)
Statistic tool for sequences with discrepancy calculation.

8.141 ql/Math/errorfunction.hpp File Reference

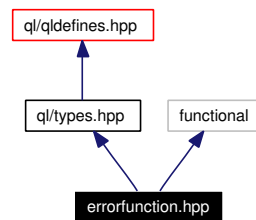
8.141.1 Detailed Description

Error function.

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for errorfunction.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ErrorFunction](#)
Error function

8.142 ql/Math/extrapolation.hpp File Reference

8.142.1 Detailed Description

class-wide extrapolation settings

```
#include <ql/qldefines.hpp>
```

Include dependency graph for extrapolation.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Extrapolator](#)
base class for classes possibly allowing extrapolation

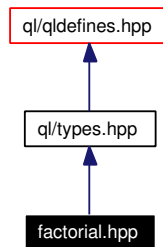
8.143 ql/Math/factorial.hpp File Reference

8.143.1 Detailed Description

Factorial numbers calculator.

```
#include <ql/types.hpp>
```

Include dependency graph for factorial.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Factorial](#)
Factorial numbers calculator

8.144 ql/Math/functional.hpp File Reference

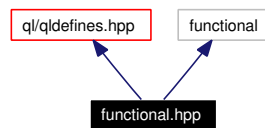
8.144.1 Detailed Description

functionals and combinators not included in the STL

```
#include <ql/qldefines.hpp>
```

```
#include <functional>
```

Include dependency graph for functional.hpp:



Namespaces

- namespace **QuantLib**

Functions

- template<class F, class R> clipped_function< F, R > **clip** (const F &f, const R &r)
- template<class F, class G> composed_function< F, G > **compose** (const F &f, const G &g)

8.145 ql/Math/gammadistribution.hpp File Reference

8.145.1 Detailed Description

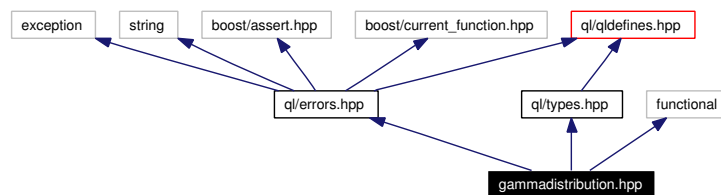
Gamma distribution.

```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for gammadistribution.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GammaFunction](#)
Gamma function class.

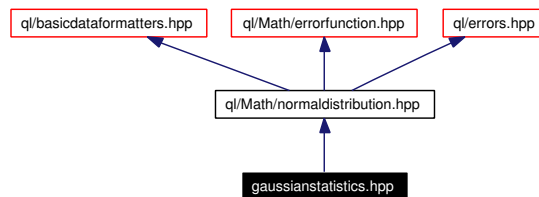
8.146 ql/Math/gaussianstatistics.hpp File Reference

8.146.1 Detailed Description

statistics tool for gaussian-assumption risk measures

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for gaussianstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GaussianStatistics](#)
Statistics tool for gaussian-assumption risk measures.
- class [StatsHolder](#)
Helper class for precomputed distributions.

8.147 ql/Math/generalstatistics.hpp File Reference

8.147.1 Detailed Description

statistics tool

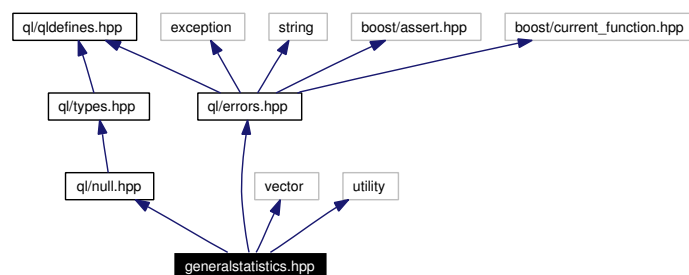
```
#include <ql/null.hpp>
```

```
#include <ql/errors.hpp>
```

```
#include <vector>
```

```
#include <utility>
```

Include dependency graph for generalstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GeneralStatistics](#)
Statistics tool.

8.148 ql/Math/incompletegamma.hpp File Reference

8.148.1 Detailed Description

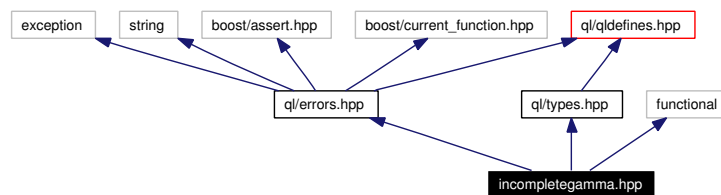
Incomplete Gamma function.

```
#include <ql/errors.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <functional>
```

Include dependency graph for incompletegamma.hpp:



Namespaces

- namespace **QuantLib**

Functions

- **Real** **incompleteGammaFunction** (**Real** a, **Real** x, **Real** accuracy=1.0e-13, **Integer** maxIteration=100)
Incomplete Gamma function.
- **Real** **incompleteGammaFunctionSeriesRepr** (**Real** a, **Real** x, **Real** accuracy=1.0e-13, **Integer** maxIteration=100)
- **Real** **incompleteGammaFunctionContinuedFractionRepr** (**Real** a, **Real** x, **Real** accuracy=1.0e-13, **Integer** maxIteration=100)

8.148.2 Function Documentation

8.148.2.1 **Real** incompleteGammaFunction (**Real** a, **Real** x, **Real** accuracy = 1.0e-13, **Integer** maxIteration = 100)

Incomplete Gamma function.

Incomplete Gamma function

The implementation of the algorithm was inspired by "Numerical Recipes in C", 2nd edition, Press, Teukolsky, Vetterling, Flannery, chapter 6

8.149 ql/Math/incrementalstatistics.hpp File Reference

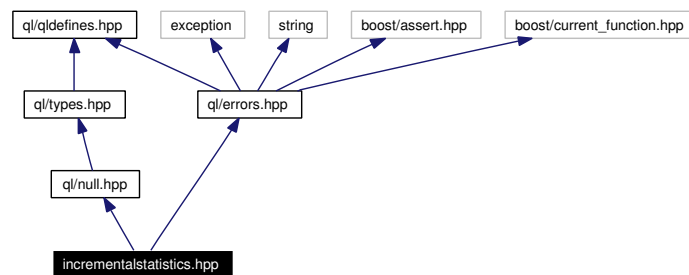
8.149.1 Detailed Description

statistics tool based on incremental accumulation

```
#include <ql/null.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for incrementalstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [IncrementalStatistics](#)
Statistics tool based on incremental accumulation.

8.150 ql/Math/interpolation.hpp File Reference

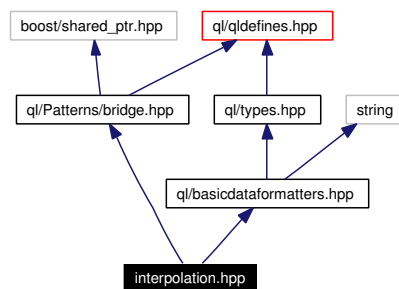
8.150.1 Detailed Description

base class for 1-D interpolations

```
#include <ql/Patterns/bridge.hpp>
```

```
#include <ql/basicdataformatters.hpp>
```

Include dependency graph for interpolation.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InterpolationImpl](#)
abstract base class for interpolation implementations
- class [Interpolation](#)
base class for 1-D interpolations.
- class [Interpolation::templateImpl](#)
basic template implementation

8.151 ql/Math/interpolation2D.hpp File Reference

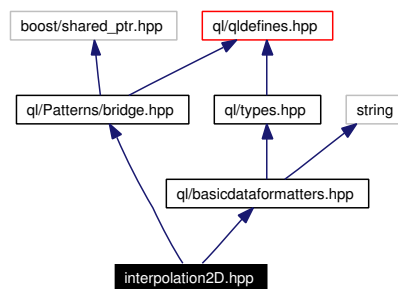
8.151.1 Detailed Description

abstract base classes for 2-D interpolations

```
#include <ql/Patterns/bridge.hpp>
```

```
#include <ql/basicdataformatters.hpp>
```

Include dependency graph for interpolation2D.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Interpolation2DImpl](#)
abstract base class for 2-D interpolation implementations
- class [Interpolation2D](#)
base class for 2-D interpolations.
- class [Interpolation2D::templateImpl](#)
basic template implementation

8.152 ql/Math/interpolationtraits.hpp File Reference

8.152.1 Detailed Description

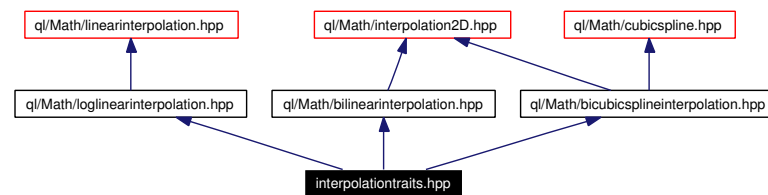
traits classes for interpolation algorithms

```
#include <ql/Math/loglinearinterpolation.hpp>
```

```
#include <ql/Math/bilinearinterpolation.hpp>
```

```
#include <ql/Math/bicubicsplineinterpolation.hpp>
```

Include dependency graph for interpolationtraits.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Linear**
Linear interpolation traits
- class **LogLinear**
Log-linear interpolation traits.
- class **Cubic**
Cubic-spline interpolation traits.

8.153 ql/Math/kronrodintegral.hpp File Reference

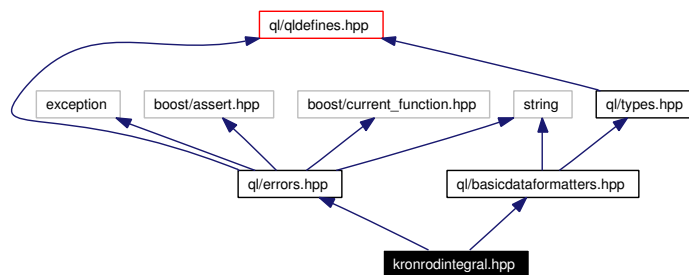
8.153.1 Detailed Description

Integral of a 1-dimensional function using the Gauss-Kronrod method.

```
#include <ql/errors.hpp>
```

```
#include <ql/basicdataformatters.hpp>
```

Include dependency graph for kronrodintegral.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **KronrodIntegral**

Integral of a 1-dimensional function using the Gauss-Kronrod method.

8.154 ql/Math/lexicographicalview.hpp File Reference

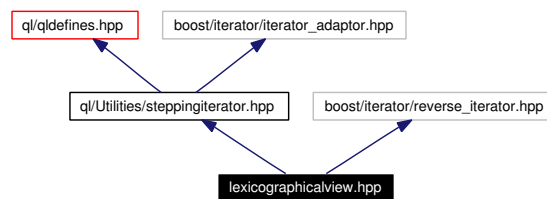
8.154.1 Detailed Description

Lexicographical 2-D view of a contiguous set of data.

```
#include <ql/Utilities/steppingiterator.hpp>
```

```
#include <boost/iterator/reverse_iterator.hpp>
```

Include dependency graph for lexicographicalview.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LexicographicalView](#)
Lexicographical 2-D view of a contiguous set of data.

8.155 ql/Math/linearinterpolation.hpp File Reference

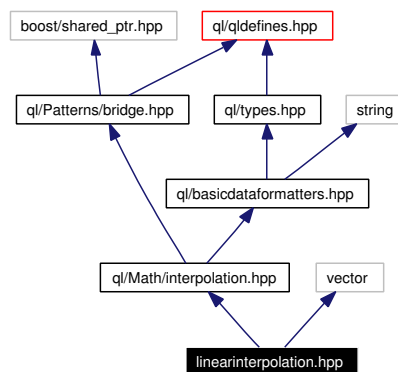
8.155.1 Detailed Description

linear interpolation between discrete points

```
#include <ql/Math/interpolation.hpp>
```

```
#include <vector>
```

Include dependency graph for linearinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [LinearInterpolation](#)
Linear interpolation between discrete points

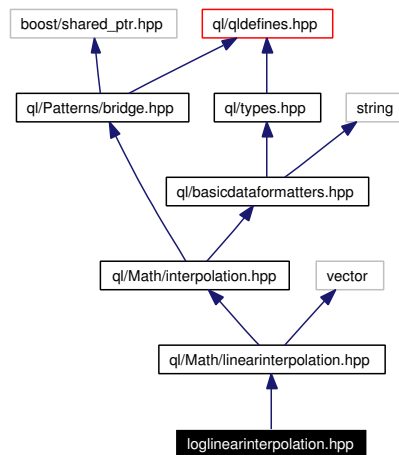
8.156 ql/Math/loglinearinterpolation.hpp File Reference

8.156.1 Detailed Description

log-linear interpolation between discrete points

```
#include <ql/Math/linearinterpolation.hpp>
```

Include dependency graph for loglinearinterpolation.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [LogLinearInterpolation](#)

8.157 ql/Math/matrix.hpp File Reference

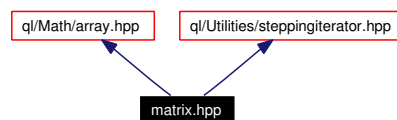
8.157.1 Detailed Description

matrix used in linear algebra.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Utilities/steppingiterator.hpp>
```

Include dependency graph for matrix.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Matrix](#)
Matrix used in linear algebra.
- class [MatrixFormatter](#)
format matrices for output

8.158 ql/Math/multicubicspline.hpp File Reference

8.158.1 Detailed Description

N-dimensional cubic spline interpolation between discrete points.

```
#include <ql/errors.hpp>
```

```
#include <ql/basicdataformatters.hpp>
```

```
#include <functional>
```

```
#include <vector>
```

Include dependency graph for multicubicspline.hpp:



Namespaces

- namespace **QuantLib**
- namespace **QuantLib::detail**

Classes

- class [MultiCubicSpline](#)

Typedefs

- typedef std::vector< std::vector< [Real](#) > > **SplineGrid**
- typedef DataTable< [Real](#) > **base_data_table**
- typedef Data< std::vector< [Real](#) >, EmptyArg > **base_data**
- typedef Point< [Real](#), EmptyArg > **base_arg_type**
- typedef Point< [Real](#), EmptyRes > **base_return_type**
- typedef Point< [Size](#), EmptyDim > **base_dimensions**
- typedef Point< base_data_table, EmptyRes > **base_output_data**
- typedef base_cubic_spline **cubic_spline_01**
- typedef n_cubic_spline< cubic_spline_01 > **cubic_spline_02**
- typedef n_cubic_spline< cubic_spline_02 > **cubic_spline_03**
- typedef n_cubic_spline< cubic_spline_03 > **cubic_spline_04**
- typedef n_cubic_spline< cubic_spline_04 > **cubic_spline_05**
- typedef n_cubic_spline< cubic_spline_05 > **cubic_spline_06**
- typedef n_cubic_spline< cubic_spline_06 > **cubic_spline_07**
- typedef n_cubic_spline< cubic_spline_07 > **cubic_spline_08**
- typedef n_cubic_spline< cubic_spline_08 > **cubic_spline_09**
- typedef n_cubic_spline< cubic_spline_09 > **cubic_spline_10**
- typedef n_cubic_spline< cubic_spline_10 > **cubic_spline_11**
- typedef n_cubic_spline< cubic_spline_11 > **cubic_spline_12**

- typedef n_cubic_spline< cubic_spline_12 > **cubic_spline_13**
- typedef n_cubic_spline< cubic_spline_13 > **cubic_spline_14**
- typedef n_cubic_spline< cubic_spline_14 > **cubic_spline_15**
- typedef base_cubic_splint **cubic_splint_01**
- typedef n_cubic_splint< cubic_splint_01 > **cubic_splint_02**
- typedef n_cubic_splint< cubic_splint_02 > **cubic_splint_03**
- typedef n_cubic_splint< cubic_splint_03 > **cubic_splint_04**
- typedef n_cubic_splint< cubic_splint_04 > **cubic_splint_05**
- typedef n_cubic_splint< cubic_splint_05 > **cubic_splint_06**
- typedef n_cubic_splint< cubic_splint_06 > **cubic_splint_07**
- typedef n_cubic_splint< cubic_splint_07 > **cubic_splint_08**
- typedef n_cubic_splint< cubic_splint_08 > **cubic_splint_09**
- typedef n_cubic_splint< cubic_splint_09 > **cubic_splint_10**
- typedef n_cubic_splint< cubic_splint_10 > **cubic_splint_11**
- typedef n_cubic_splint< cubic_splint_11 > **cubic_splint_12**
- typedef n_cubic_splint< cubic_splint_12 > **cubic_splint_13**
- typedef n_cubic_splint< cubic_splint_13 > **cubic_splint_14**
- typedef n_cubic_splint< cubic_splint_14 > **cubic_splint_15**
- typedef detail::SplineGrid **SplineGrid**

8.159 ql/Math/normaldistribution.hpp File Reference

8.159.1 Detailed Description

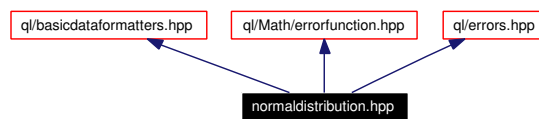
normal, cumulative and inverse cumulative distributions

```
#include <ql/basicdataformatters.hpp>
```

```
#include <ql/Math/errorfunction.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for normaldistribution.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [NormalDistribution](#)
Normal distribution function.
- class [CumulativeNormalDistribution](#)
Cumulative normal distribution function.
- class [InverseCumulativeNormal](#)
Inverse cumulative normal distribution function.
- class [MoroInverseCumulativeNormal](#)
Moro Inverse cumulative normal distribution class.

Typedefs

- typedef NormalDistribution **GaussianDistribution**
- typedef InverseCumulativeNormal **InvCumulativeNormalDistribution**

8.160 ql/Math/poissondistribution.hpp File Reference

8.160.1 Detailed Description

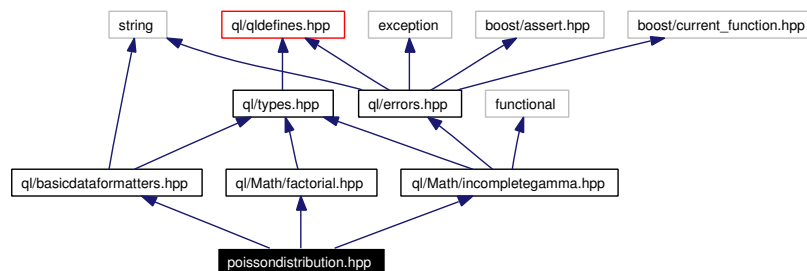
Poisson distribution.

```
#include <ql/basicdataformatters.hpp>
```

```
#include <ql/Math/factorial.hpp>
```

```
#include <ql/Math/incompletegamma.hpp>
```

Include dependency graph for poissondistribution.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PoissonDistribution](#)
Normal distribution function.
- class [CumulativePoissonDistribution](#)
Cumulative Poisson distribution function.
- class [InverseCumulativePoisson](#)
Inverse cumulative Poisson distribution function.

8.161 ql/Math/primenumbers.hpp File Reference

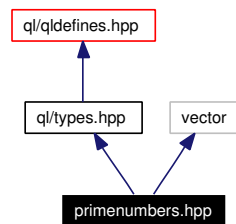
8.161.1 Detailed Description

Prime numbers calculator.

```
#include <ql/types.hpp>
```

```
#include <vector>
```

Include dependency graph for primenumbers.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PrimeNumbers](#)
Prime numbers calculator.

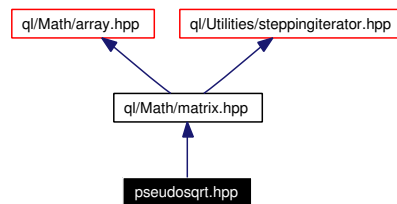
8.162 ql/Math/pseudosqrt.hpp File Reference

8.162.1 Detailed Description

pseudo square root of a real symmetric matrix

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for pseudosqrt.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct [SalvagingAlgorithm](#)
algorithm used for matricial pseudo square root

8.163 ql/Math/riskstatistics.hpp File Reference

8.163.1 Detailed Description

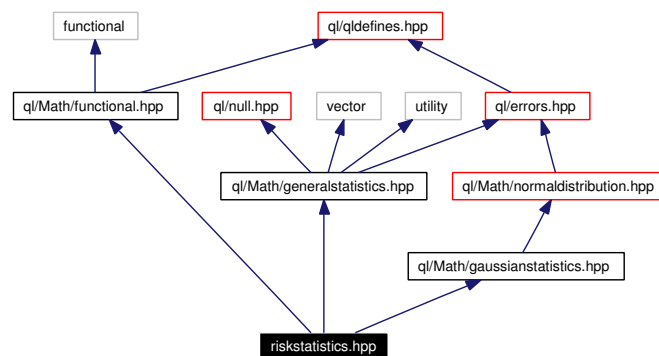
empirical-distribution risk measures

```
#include <ql/Math/functional.hpp>
```

```
#include <ql/Math/generalstatistics.hpp>
```

```
#include <ql/Math/gaussianstatistics.hpp>
```

Include dependency graph for riskstatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GenericRiskStatistics](#)
empirical-distribution risk measures

Typedefs

- typedef `GaussianStatistics< GenericRiskStatistics< GeneralStatistics > >` [RiskStatistics](#)
default risk measures tool

8.163.2 Typedef Documentation

8.163.2.1 typedef `GaussianStatistics<GenericRiskStatistics<GeneralStatistics> >` `RiskStatistics`

default risk measures tool

Tests

the correctness of the returned values is tested by checking them against numerical calculations.

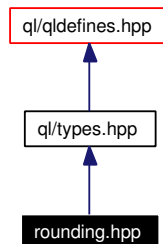
8.164 ql/Math/rounding.hpp File Reference

8.164.1 Detailed Description

Rounding implementation.

```
#include <ql/types.hpp>
```

Include dependency graph for rounding.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Rounding](#)
basic rounding class
- class [UpRounding](#)
Up-rounding.
- class [DownRounding](#)
Down-rounding.
- class [ClosestRounding](#)
Closest rounding.
- class [CeilingTruncation](#)
Ceiling truncation.
- class [FloorTruncation](#)
Floor truncation.

8.165 ql/Math/segmentintegral.hpp File Reference

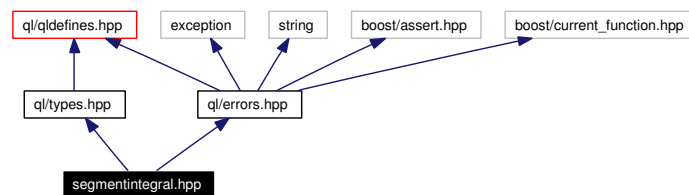
8.165.1 Detailed Description

Integral of a one-dimensional function.

```
#include <ql/types.hpp>
```

```
#include <ql/errors.hpp>
```

Include dependency graph for segmentintegral.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SegmentIntegral](#)
Integral of a one-dimensional function.

8.166 ql/Math/sequencestatistics.hpp File Reference

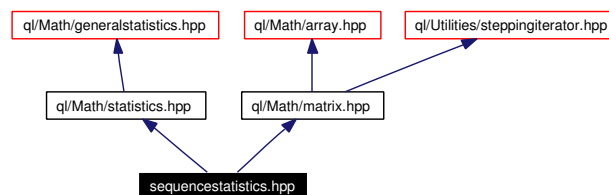
8.166.1 Detailed Description

Statistics tools for sequence (vector, list, array) samples.

```
#include <ql/Math/statistics.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for sequencestatistics.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SequenceStatistics](#)
Statistics analysis of N-dimensional (sequence) data.

Defines

- #define **DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID**(METHOD)
- #define **DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE**(METHOD)

8.166.2 Define Documentation

8.166.2.1 #define **DEFINE_SEQUENCE_STAT_CONST_METHOD_VOID**(METHOD)

Value:

```

template <class Stat> \
    std::vector<Real> \
    SequenceStatistics<Stat>::METHOD() const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(); \
        return results_; \
    }

```

8.166.2.2 #define DEFINE_SEQUENCE_STAT_CONST_METHOD_DOUBLE(METHOD)**Value:**

```
template <class Stat> \
    std::vector<Real> \
    SequenceStatistics<Stat>::METHOD(Real x) const { \
        for (Size i=0; i<dimension_; i++) \
            results_[i] = stats_[i].METHOD(x); \
        return results_; \
    }
```

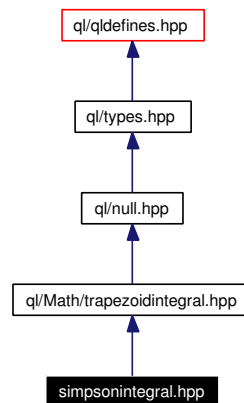
8.167 ql/Math/simpsonintegral.hpp File Reference

8.167.1 Detailed Description

integral of a one-dimensional function

```
#include <ql/Math/trapezoidintegral.hpp>
```

Include dependency graph for `simpsonintegral.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [SimpsonIntegral](#)
Integral of a one-dimensional function.

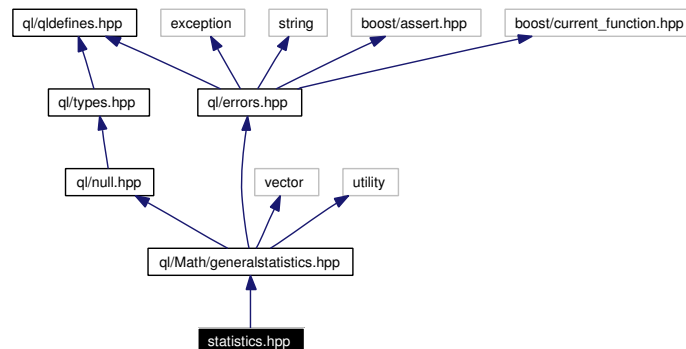
8.168 ql/Math/statistics.hpp File Reference

8.168.1 Detailed Description

statistics tool with risk measures

```
#include <ql/Math/generalstatistics.hpp>
```

Include dependency graph for statistics.hpp:



Namespaces

- namespace `QuantLib`

Typedefs

- typedef `GeneralStatistics` [Statistics](#)
default statistics tool

8.168.2 Typedef Documentation

8.168.2.1 typedef GeneralStatistics Statistics

default statistics tool

Tests

the correctness of the returned values is tested by checking them against numerical calculations.

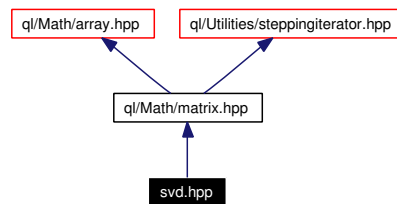
8.169 ql/Math/svd.hpp File Reference

8.169.1 Detailed Description

singular value decomposition

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for svd.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **SVD**
Singular value decomposition.

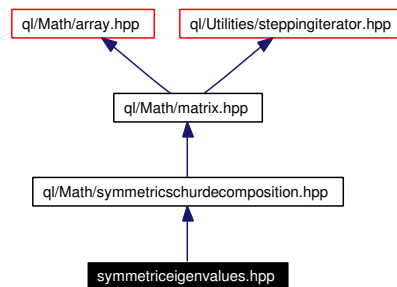
8.170 ql/Math/symmetriceigenvalues.hpp File Reference

8.170.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

```
#include <ql/Math/symmetricschurdecomposition.hpp>
```

Include dependency graph for symmetriceigenvalues.hpp:



Namespaces

- namespace **QuantLib**

Functions

- Disposable< Array > **SymmetricEigenvalues** (Matrix &s)
- Disposable< Matrix > **SymmetricEigenvectors** (Matrix &s)

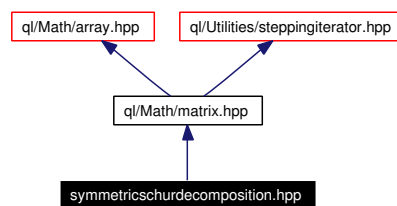
8.171 ql/Math/symmetricschurdecomposition.hpp File Reference

8.171.1 Detailed Description

Eigenvalues / eigenvectors of a real symmetric matrix.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for symmetricschurdecomposition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SymmetricSchurDecomposition](#)
symmetric threshold Jacobi algorithm.

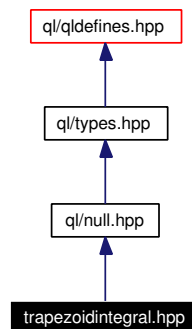
8.172 ql/Math/trapezoidintegral.hpp File Reference

8.172.1 Detailed Description

integral of a one-dimensional function

```
#include <ql/null.hpp>
```

Include dependency graph for trapezoidintegral.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TrapezoidIntegral](#)
Integral of a one-dimensional function.

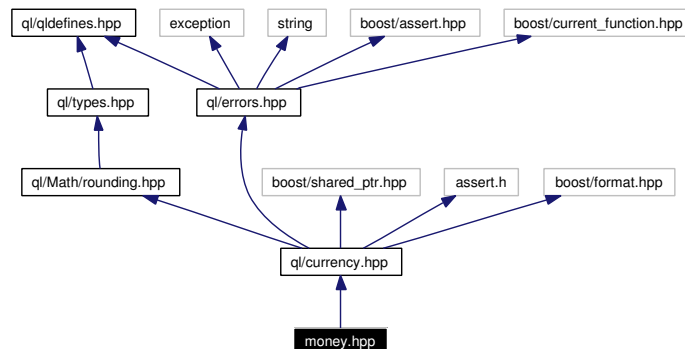
8.173 ql/money.hpp File Reference

8.173.1 Detailed Description

cash amount in a given currency

```
#include <ql/currency.hpp>
```

Include dependency graph for money.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **Money**
amount of cash
- class **MoneyFormatter**
format money for output

8.174 ql/MonteCarlo/brownianbridge.hpp File Reference

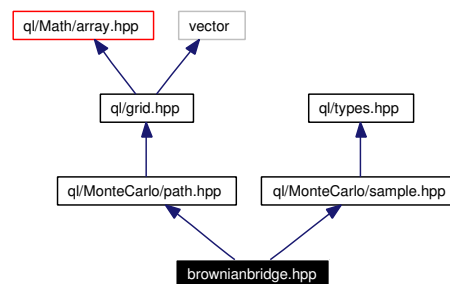
8.174.1 Detailed Description

Browian bridge.

```
#include <ql/MonteCarlo/path.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for brownianbridge.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BrownianBridge**
Builds Wiener process paths using Gaussian variates.

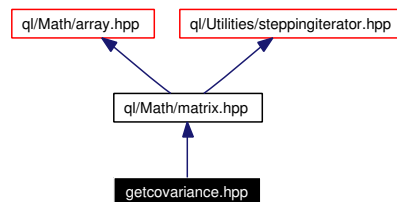
8.175 ql/MonteCarlo/getcovariance.hpp File Reference

8.175.1 Detailed Description

Covariance matrix calculation.

```
#include <ql/Math/matrix.hpp>
```

Include dependency graph for getcovariance.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `CovarianceDecomposition`

Functions

- `template<class DataIterator> Disposable< Matrix > getCovariance (DataIterator volBegin, DataIterator volEnd, const Matrix &corr, Real tolerance=1.0e-12)`

8.175.2 Function Documentation

8.175.2.1 `Disposable<Matrix> getCovariance (DataIterator volBegin, DataIterator volEnd, const Matrix & corr, Real tolerance = 1.0e-12)`

Combines the correlation matrix and the vector of volatilities to return the covariance matrix.

Note that only the symmetric part of the correlation matrix is used. Also it is assumed that the diagonal member of the correlation matrix equals one.

Precondition:

The correlation matrix must be symmetric with the diagonal members equal to one.

Tests

tested on know values and cross checked with `CovarianceDecomposition`

8.176 ql/MonteCarlo/mctraits.hpp File Reference

8.176.1 Detailed Description

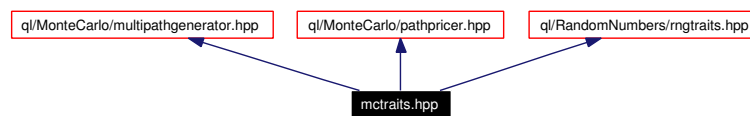
Monte Carlo policies.

```
#include <ql/MonteCarlo/multipathgenerator.hpp>
```

```
#include <ql/MonteCarlo/pathpricer.hpp>
```

```
#include <ql/RandomNumbers/rngtraits.hpp>
```

Include dependency graph for mctraits.hpp:



Namespaces

- namespace **QuantLib**

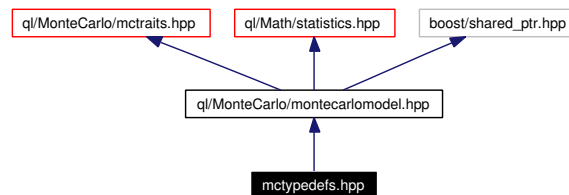
8.177 ql/MonteCarlo/mctypedefs.hpp File Reference

8.177.1 Detailed Description

Default choices for template instantiations.

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

Include dependency graph for mctypedefs.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef `MonteCarloModel< SingleAsset< PseudoRandom > >` [OneFactorMonteCarloOption](#)
default choice for one-factor Monte Carlo model.
- typedef `MonteCarloModel< MultiAsset< PseudoRandom > >` [MultiFactorMonteCarloOption](#)
default choice for multi-factor Monte Carlo model.

8.178 ql/MonteCarlo/montecarlomodel.hpp File Reference

8.178.1 Detailed Description

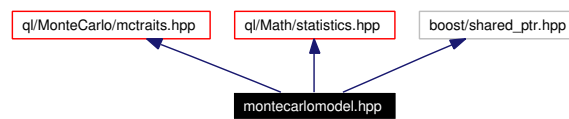
General purpose Monte Carlo model.

```
#include <ql/MonteCarlo/mctraits.hpp>
```

```
#include <ql/Math/statistics.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for montecarlomodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MonteCarloModel](#)
General purpose Monte Carlo model for path samples.

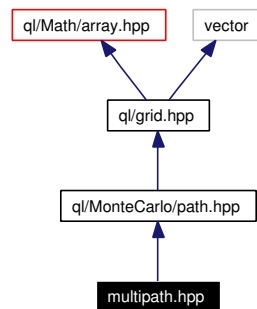
8.179 ql/MonteCarlo/multipath.hpp File Reference

8.179.1 Detailed Description

Correlated multiple asset paths.

```
#include <ql/MonteCarlo/path.hpp>
```

Include dependency graph for multipath.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MultiPath](#)
Correlated multiple asset paths.

8.180 ql/MonteCarlo/multipathgenerator.hpp File Reference

8.180.1 Detailed Description

Generates a multi path from a random-array generator.

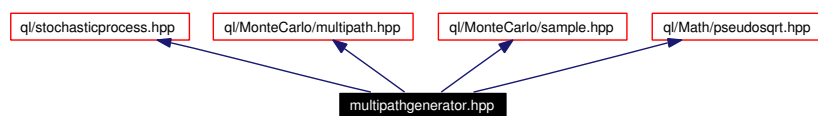
```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/MonteCarlo/multipath.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <ql/Math/pseudosqrt.hpp>
```

Include dependency graph for multipathgenerator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MultiPathGenerator](#)

Generates a multipath from a random number generator.

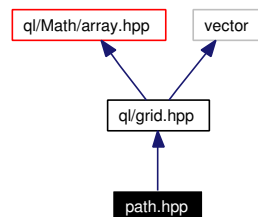
8.181 ql/MonteCarlo/path.hpp File Reference

8.181.1 Detailed Description

single factor random walk

```
#include <ql/grid.hpp>
```

Include dependency graph for path.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `Path`

8.182 ql/MonteCarlo/pathgenerator.hpp File Reference

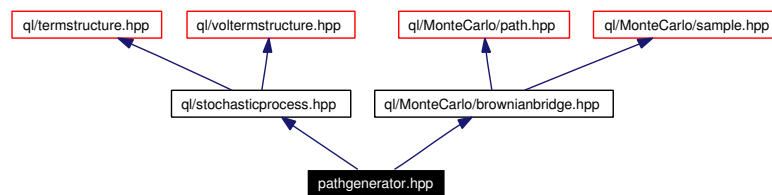
8.182.1 Detailed Description

Generates random paths using a sequence generator.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/MonteCarlo/brownianbridge.hpp>
```

Include dependency graph for pathgenerator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PathGenerator](#)
Generates random paths using a sequence generator.

8.183 ql/MonteCarlo/pathpricer.hpp File Reference

8.183.1 Detailed Description

base class for single-path pricers

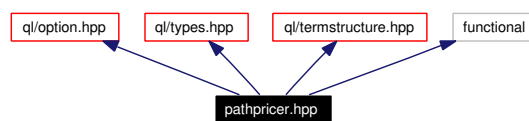
```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <functional>
```

Include dependency graph for pathpricer.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PathPricer](#)
base class for path pricers

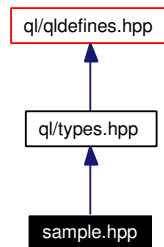
8.184 ql/MonteCarlo/sample.hpp File Reference

8.184.1 Detailed Description

weighted sample

```
#include <ql/types.hpp>
```

Include dependency graph for sample.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct [Sample](#)
weighted sample

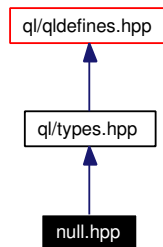
8.185 ql/null.hpp File Reference

8.185.1 Detailed Description

null values

```
#include <ql/types.hpp>
```

Include dependency graph for null.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Null](#)
template class providing a null value for a given type.

8.186 ql/numericalmethod.hpp File Reference

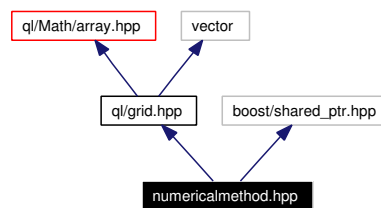
8.186.1 Detailed Description

Numerical method class.

```
#include <ql/grid.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for numericalmethod.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [NumericalMethod](#)
Numerical method (tree, finite-differences) base class.

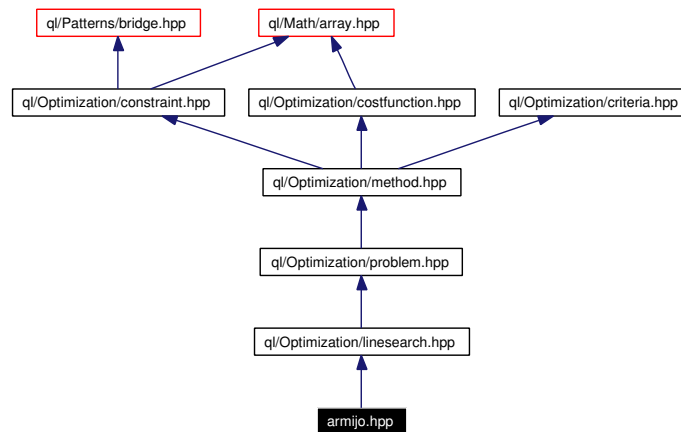
8.187 ql/Optimization/armijo.hpp File Reference

8.187.1 Detailed Description

Armijo line-search class.

```
#include <ql/Optimization/linesearch.hpp>
```

Include dependency graph for armijo.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ArmijoLineSearch](#)
Armijo line search.

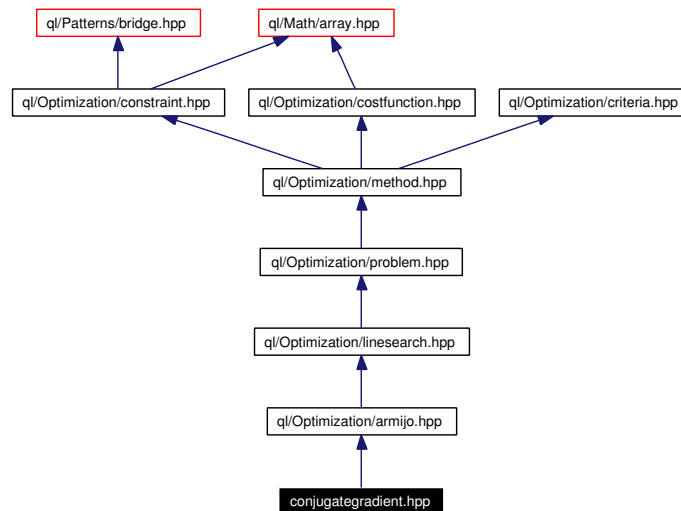
8.188 ql/Optimization/conjugategradient.hpp File Reference

8.188.1 Detailed Description

Conjugate gradient optimization method.

```
#include <ql/Optimization/armijo.hpp>
```

Include dependency graph for conjugategradient.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ConjugateGradient](#)
Multi-dimensional Conjugate Gradient class.

8.189 ql/Optimization/constraint.hpp File Reference

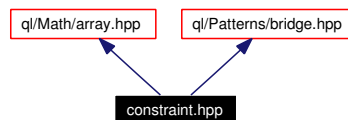
8.189.1 Detailed Description

Abstract constraint class.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/Patterns/bridge.hpp>
```

Include dependency graph for constraint.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ConstraintImpl](#)
Base class for constraint implementations.
- class [Constraint](#)
Base constraint class.
- class [NoConstraint](#)
No constraint.
- class [PositiveConstraint](#)
Constraint imposing positivity to all arguments
- class [BoundaryConstraint](#)
Constraint imposing all arguments to be in [low,high]
- class [CompositeConstraint](#)
Constraint enforcing both given sub-constraints

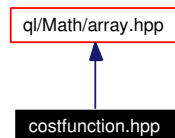
8.190 ql/Optimization/costfunction.hpp File Reference

8.190.1 Detailed Description

Optimization cost function class.

```
#include <ql/Math/array.hpp>
```

Include dependency graph for costfunction.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CostFunction](#)
Cost function abstract class for optimization problem.

8.191 ql/Optimization/criteria.hpp File Reference

8.191.1 Detailed Description

Optimization criteria class.

Namespaces

- namespace **QuantLib**

Classes

- class [EndCriteria](#)
Criteria to end optimization process.

8.192 ql/Optimization/leastsquare.hpp File Reference

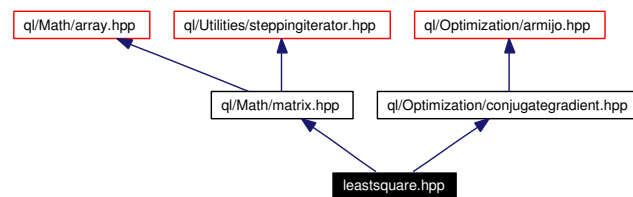
8.192.1 Detailed Description

Least square cost function.

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Optimization/conjugategradient.hpp>
```

Include dependency graph for leastsquare.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LeastSquareProblem](#)
Base class for least square problem.
- class [LeastSquareFunction](#)
Cost function for least-square problems.
- class [NonLinearLeastSquare](#)
Non-linear least-square method.

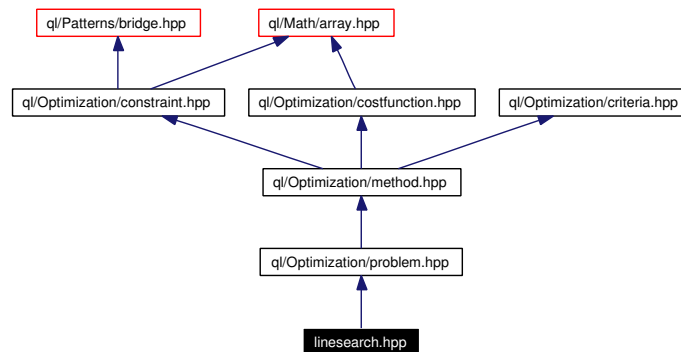
8.193 ql/Optimization/linesearch.hpp File Reference

8.193.1 Detailed Description

Line search abstract class.

```
#include <ql/Optimization/problem.hpp>
```

Include dependency graph for linesearch.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LineSearch](#)
Base class for line search.

8.194 ql/Optimization/method.hpp File Reference

8.194.1 Detailed Description

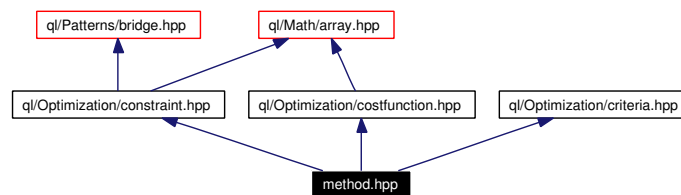
Abstract optimization method class.

```
#include <ql/Optimization/constraint.hpp>
```

```
#include <ql/Optimization/costfunction.hpp>
```

```
#include <ql/Optimization/criteria.hpp>
```

Include dependency graph for method.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OptimizationMethod](#)
Abstract class for constrained optimization method.

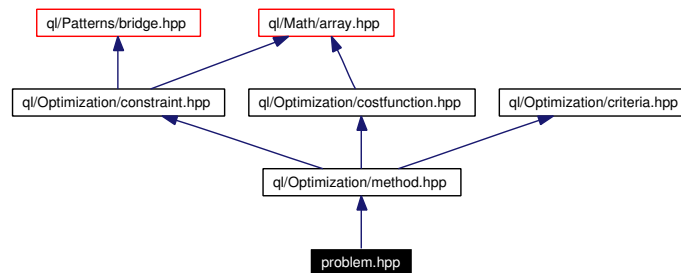
8.195 ql/Optimization/problem.hpp File Reference

8.195.1 Detailed Description

Abstract optimization class.

```
#include <ql/Optimization/method.hpp>
```

Include dependency graph for problem.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Problem](#)
Constrained optimization problem.

8.196 ql/Optimization/simplex.hpp File Reference

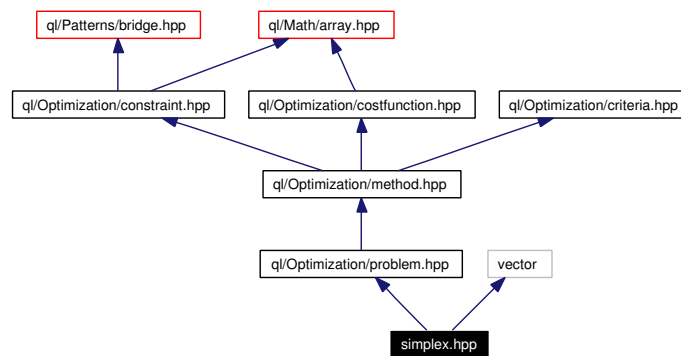
8.196.1 Detailed Description

Simplex optimization method.

```
#include <ql/Optimization/problem.hpp>
```

```
#include <vector>
```

Include dependency graph for simplex.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Simplex](#)
Multi-dimensional simplex class.

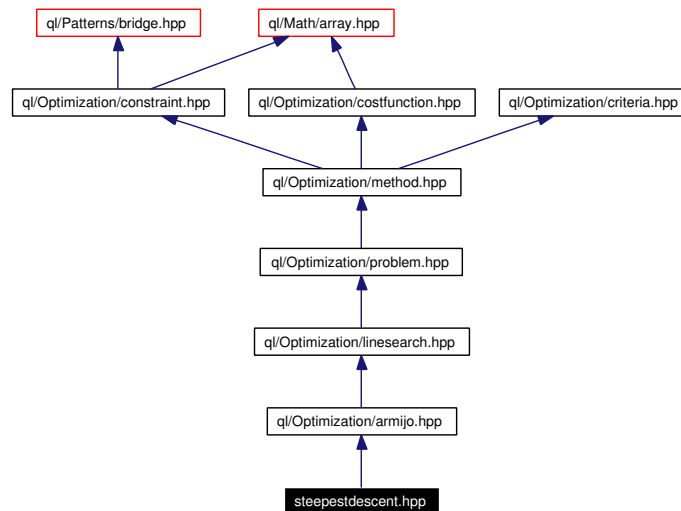
8.197 ql/Optimization/steepestdescent.hpp File Reference

8.197.1 Detailed Description

Steepest descent optimization method.

```
#include <ql/Optimization/armijo.hpp>
```

Include dependency graph for steepestdescent.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SteepestDescent](#)
Multi-dimensional steepest-descent class.

8.198 ql/option.hpp File Reference

8.198.1 Detailed Description

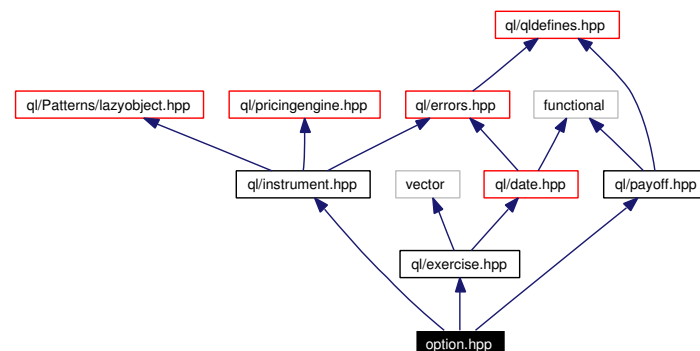
Base option class.

```
#include <ql/instrument.hpp>
```

```
#include <ql/payoff.hpp>
```

```
#include <ql/exercise.hpp>
```

Include dependency graph for option.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `Option`
base option class
- class `Option::arguments`
- class `Greeks`
additional option results
- class `MoreGreeks`
more additional option results
- class `OptionTypeFormatter`
format option type for output

8.199 ql/Patterns/bridge.hpp File Reference

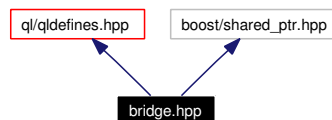
8.199.1 Detailed Description

bridge pattern (a.k.a. handle-body idiom)

```
#include <ql/qldefines.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for bridge.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Bridge](#)
The Bridge pattern made explicit.

8.200 ql/Patterns/composite.hpp File Reference

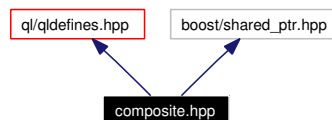
8.200.1 Detailed Description

composite pattern

```
#include <ql/qldefines.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for composite.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Composite](#)
Composite pattern.

8.201 ql/Patterns/curiouslyrecurring.hpp File Reference

8.201.1 Detailed Description

Curiously recurring template pattern.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for curiouslyrecurring.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CuriouslyRecurringTemplate](#)
Support for the curiously recurring template pattern.

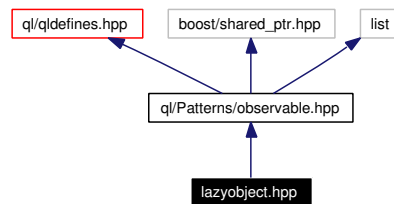
8.202 ql/Patterns/lazyobject.hpp File Reference

8.202.1 Detailed Description

framework for calculation on demand and result caching

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for lazyobject.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LazyObject](#)

Framework for calculation on demand and result caching.

8.203 ql/Patterns/observable.hpp File Reference

8.203.1 Detailed Description

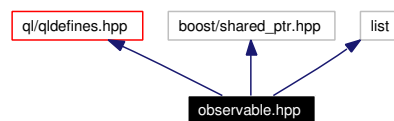
observer/observable pattern

```
#include <ql/qldefines.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

```
#include <list>
```

Include dependency graph for observable.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Observable](#)
Object that notifies its changes to a set of observables.
- class [Observer](#)
Object that gets notified when a given observable changes.

8.204 ql/Patterns/singleton.hpp File Reference

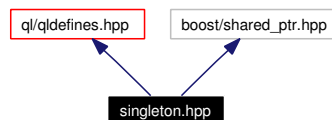
8.204.1 Detailed Description

basic support for the singleton pattern

```
#include <ql/qldefines.hpp>
```

```
#include <boost/shared_ptr.hpp>
```

Include dependency graph for singleton.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Singleton](#)
Basic support for the singleton pattern.

8.205 ql/Patterns/visitor.hpp File Reference

8.205.1 Detailed Description

degenerate base class for the Acyclic Visitor pattern

```
#include <ql/qldefines.hpp>
```

Include dependency graph for visitor.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AcyclicVisitor](#)
degenerate base class for the Acyclic Visitor pattern
- class [Visitor](#)
Visitor for a specific class

8.206 ql/payoff.hpp File Reference

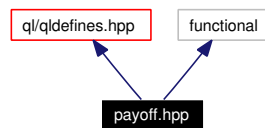
8.206.1 Detailed Description

Option payoff classes.

```
#include <ql/qldefines.hpp>
```

```
#include <functional>
```

Include dependency graph for payoff.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Payoff](#)
Base class for option payoffs.

8.207 ql/Pricers/discretegeometricapo.hpp File Reference

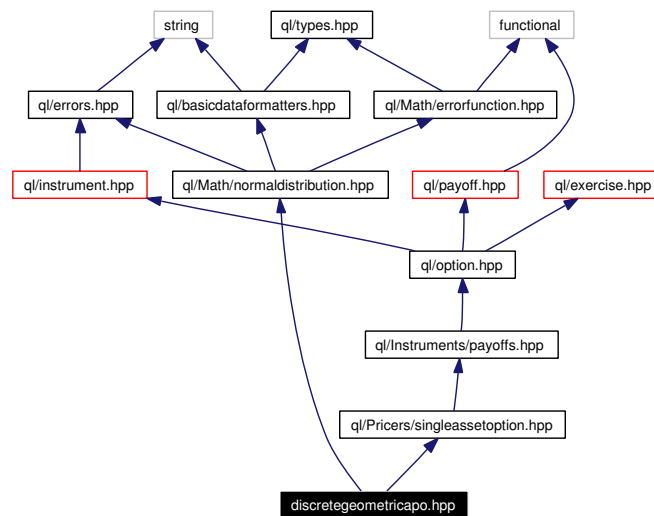
8.207.1 Detailed Description

Discrete Geometric Average Price Option.

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for discretegeometricapo.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **DiscreteGeometricAPO**
Discrete geometric average-price Asian option (European style).

8.208 ql/Pricers/discretegeometricaso.hpp File Reference

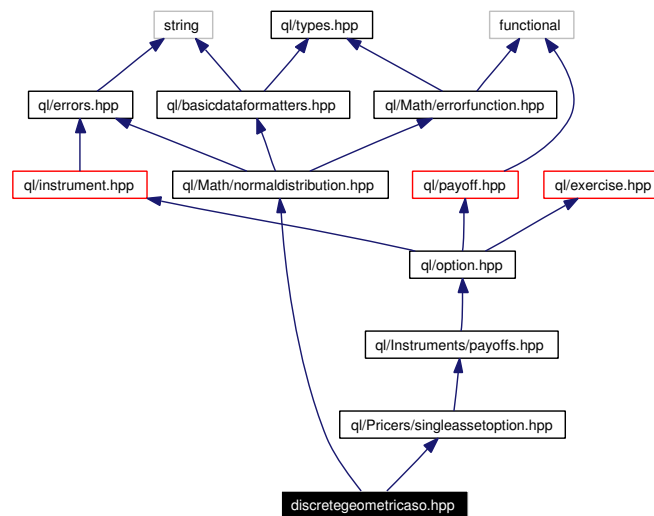
8.208.1 Detailed Description

Discrete Geometric Average Strike Option.

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for discretegeometricaso.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DiscreteGeometricASO](#)
Discrete geometric average-strike Asian option (European style).

8.209 ql/Pricers/fdamericanoption.hpp File Reference

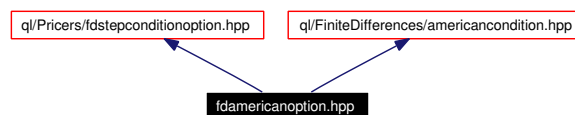
8.209.1 Detailed Description

american option

```
#include <ql/Pricers/fdstepconditionoption.hpp>
```

```
#include <ql/FiniteDifferences/americancondition.hpp>
```

Include dependency graph for fdamericanoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FdAmericanOption](#)
American option.

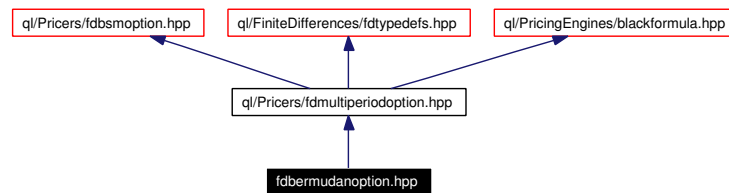
8.210 ql/Pricers/fdbermudanoption.hpp File Reference

8.210.1 Detailed Description

finite-difference evaluation of Bermudan option

```
#include <ql/Pricers/fdmultipleriodoption.hpp>
```

Include dependency graph for fdbermudanoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **FdBermudanOption**
Bermudan option.

8.211 ql/Pricers/fdbsmoption.hpp File Reference

8.211.1 Detailed Description

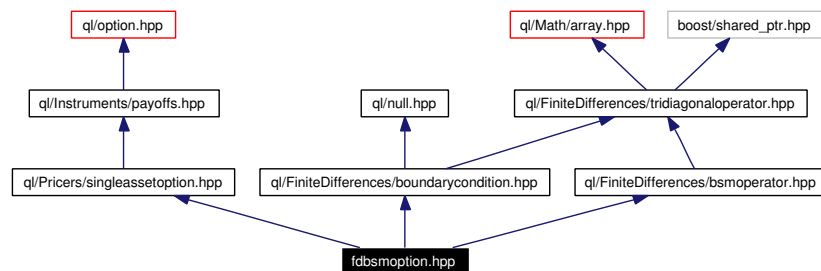
common code for numerical option evaluation

```
#include <ql/Pricers/singleassetoption.hpp>
```

```
#include <ql/FiniteDifferences/bsmoperator.hpp>
```

```
#include <ql/FiniteDifferences/boundarycondition.hpp>
```

Include dependency graph for fdbsmoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FdBsmOption](#)
Black-Scholes-Merton option priced numerically.

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.

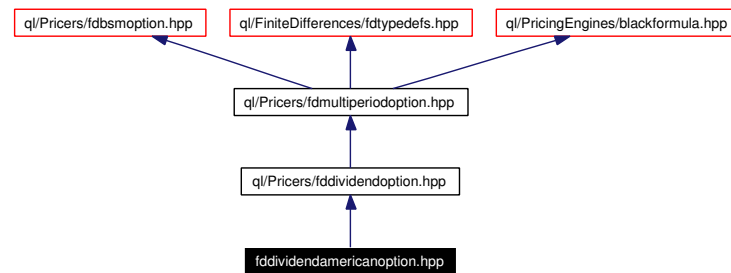
8.212 ql/Pricers/fddividendamericanoption.hpp File Reference

8.212.1 Detailed Description

american option with discrete deterministic dividends

```
#include <ql/Pricers/fddividendoption.hpp>
```

Include dependency graph for fddividendamericanoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FdDividendAmericanOption](#)
American option with discrete dividends.

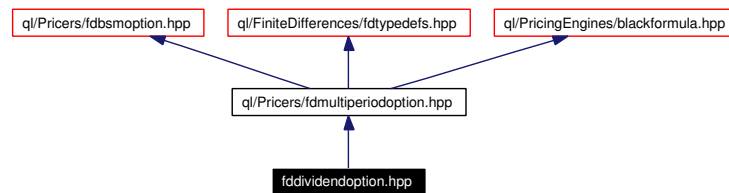
8.213 ql/Pricers/fddividendoption.hpp File Reference

8.213.1 Detailed Description

base class for option with dividends

```
#include <ql/Pricers/fdmultiperiodoption.hpp>
```

Include dependency graph for fddividendoption.hpp:



Namespaces

- namespace **QuantLib**

8.214 ql/Pricers/fddividendshoutoption.hpp File Reference

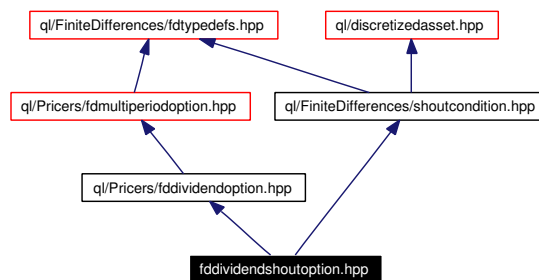
8.214.1 Detailed Description

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**

Classes

- class [FdDividendShoutOption](#)
Shout option with dividends.

8.215 ql/Pricers/fdeuropean.hpp File Reference

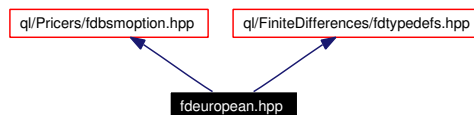
8.215.1 Detailed Description

Example of European option calculated using finite differences.

```
#include <ql/Pricers/fdbsoption.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

Include dependency graph for fdeuropean.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FdEuropean](#)

Example of European option calculated using finite differences.

8.216 ql/Pricers/fdmultiperiodoption.hpp File Reference

8.216.1 Detailed Description

base class for option with events happening at different periods

```
#include <ql/Pricers/fdbsmoption.hpp>
```

```
#include <ql/FiniteDifferences/fdtypedefs.hpp>
```

```
#include <ql/PricingEngines/blackformula.hpp>
```

Include dependency graph for fdmultiperiodoption.hpp:



Namespaces

- namespace **QuantLib**

8.217 ql/Pricers/fdshoutoption.hpp File Reference

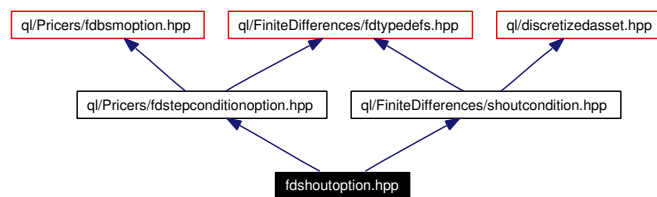
8.217.1 Detailed Description

shout option

```
#include <ql/Pricers/fdstepconditionoption.hpp>
```

```
#include <ql/FiniteDifferences/shoutcondition.hpp>
```

Include dependency graph for fdshoutoption.hpp:



Namespaces

- namespace **QuantLib**

8.218 ql/Pricers/fdstepconditionoption.hpp File Reference

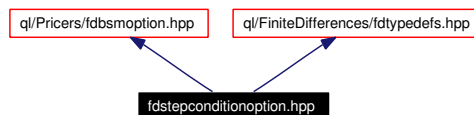
8.218.1 Detailed Description

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**

Classes

- class [FdStepConditionOption](#)
option executing additional code at each time step

8.219 ql/Pricers/mccliquestoption.hpp File Reference

8.219.1 Detailed Description

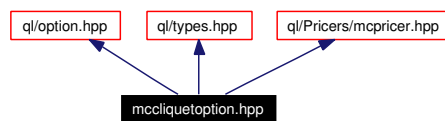
Cliquet option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mccliquestoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [McCliquetOption](#)
simple example of Monte Carlo pricer

8.220 ql/Pricers/mcdiscretearithmeticapo.hpp File Reference

8.220.1 Detailed Description

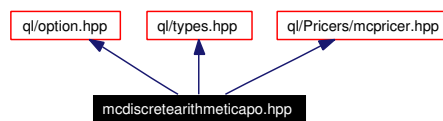
Discrete Arithmetic Average Price Option.

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mcdiscretearithmeticapo.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [McDiscreteArithmeticAPO](#)
example of Monte Carlo pricer using a control variate

8.221 ql/Pricers/mcdiscretearithmeticso.hpp File Reference

8.221.1 Detailed Description

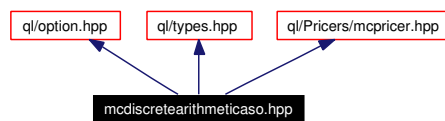
Discrete Arithmetic Average Strike Option.

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mcdiscretearithmeticso.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [McDiscreteArithmeticASO](#)
example of Monte Carlo pricer using a control variate.

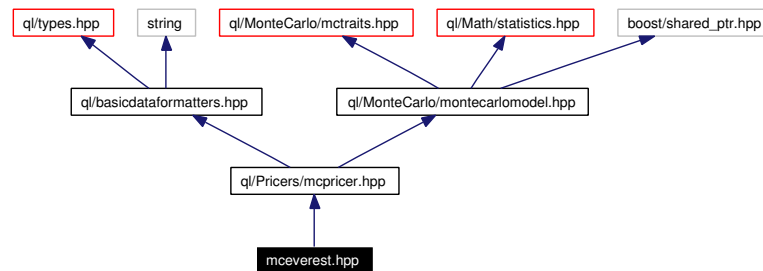
8.222 ql/Pricers/mceverest.hpp File Reference

8.222.1 Detailed Description

Everest-type option pricer

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mceverest.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McEverest**
Everest-type option pricer.

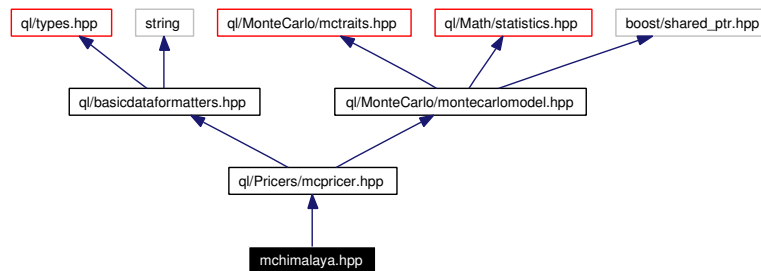
8.223 ql/Pricers/mchimalaya.hpp File Reference

8.223.1 Detailed Description

Himalayan-type option pricer.

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mchimalaya.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McHimalaya**
Himalayan-type option pricer.

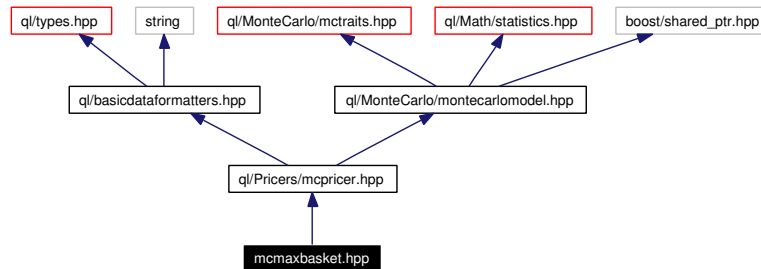
8.224 ql/Pricers/mcmaxbasket.hpp File Reference

8.224.1 Detailed Description

Max Basket Monte Carlo pricer.

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mcmaxbasket.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McMaxBasket**
simple example of multi-factor Monte Carlo pricer

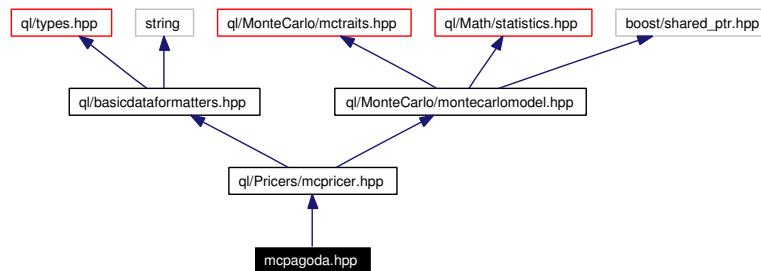
8.225 ql/Pricers/mcpagoda.hpp File Reference

8.225.1 Detailed Description

Roofed multi asset Asian option.

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mcpagoda.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McPagoda**
roofed Asian option

8.226 ql/Pricers/mcperformanceoption.hpp File Reference

8.226.1 Detailed Description

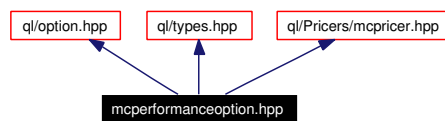
Performance option priced with Monte Carlo simulation.

```
#include <ql/option.hpp>
```

```
#include <ql/types.hpp>
```

```
#include <ql/Pricers/mcpricer.hpp>
```

Include dependency graph for mcperformanceoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [McPerformanceOption](#)
Performance option computed using Monte Carlo simulation.

8.227 ql/Pricers/mcpricer.hpp File Reference

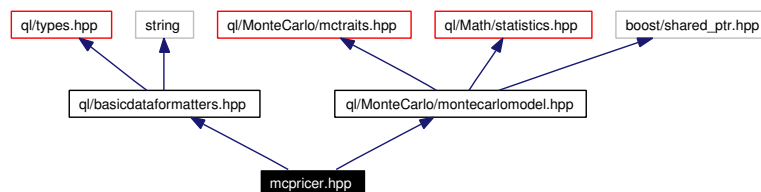
8.227.1 Detailed Description

base class for Monte Carlo pricers

```
#include <ql/basicdataformatters.hpp>
```

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

Include dependency graph for mcpricer.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **McPricer**
base class for Monte Carlo pricers

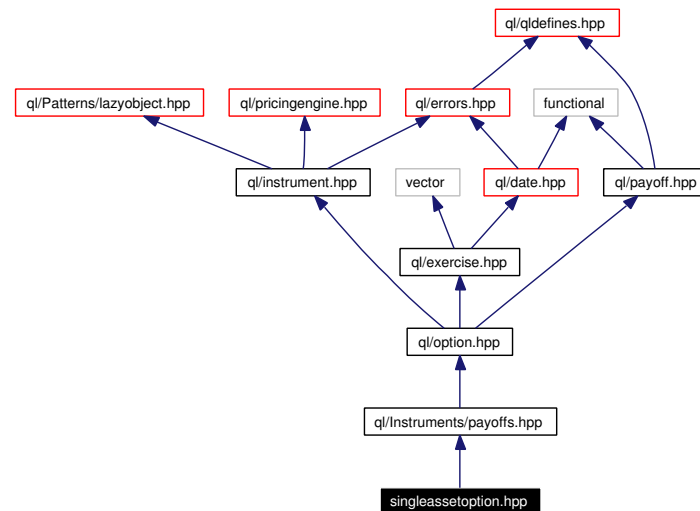
8.228 ql/Pricers/singleassetoption.hpp File Reference

8.228.1 Detailed Description

common code for option evaluation

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for singleassetoption.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SingleAssetOption](#)
Black-Scholes-Merton option.

8.229 ql/pricingengine.hpp File Reference

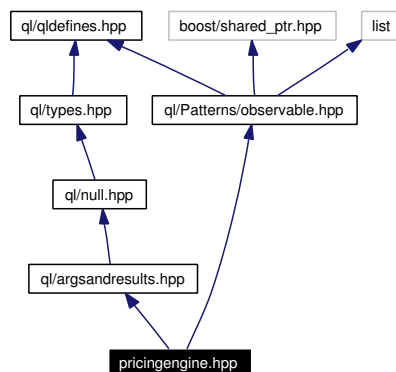
8.229.1 Detailed Description

Base class for pricing engines.

```
#include <ql/argsandresults.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for pricingengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PricingEngine](#)
interface for pricing engines
- class [GenericEngine](#)
template base class for option pricing engines

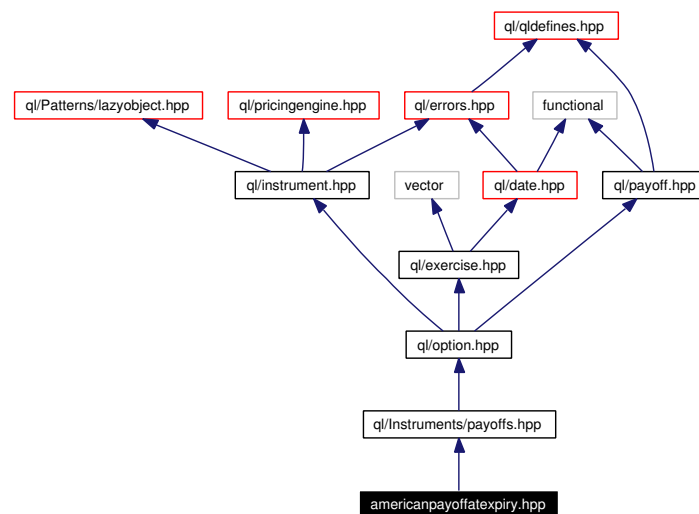
8.230 ql/PricingEngines/americanpayoffatexpiry.hpp File Reference

8.230.1 Detailed Description

Analytical formulae for american exercise with payoff at expiry.

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for americanpayoffatexpiry.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `AmericanPayoffAtExpiry`

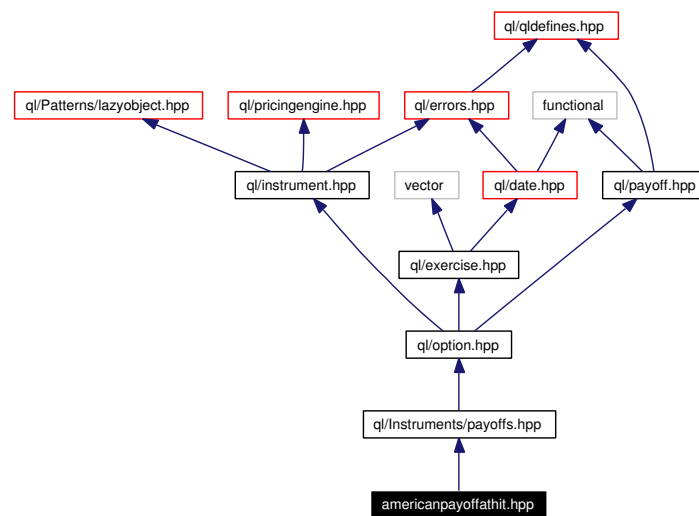
8.231 ql/PricingEngines/americanpayoffathit.hpp File Reference

8.231.1 Detailed Description

Analytical formulae for american exercise with payoff at hit.

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for americanpayoffathit.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AmericanPayoffAtHit](#)

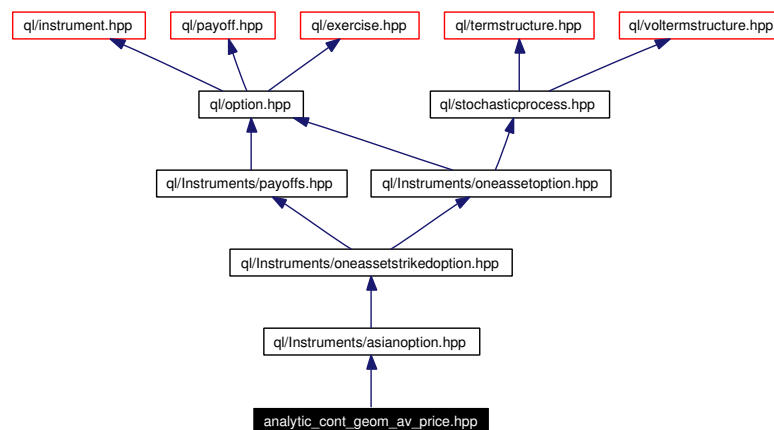
8.232 ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp File Reference

8.232.1 Detailed Description

Analytic engine for continuous geometric average price Asian.

```
#include <ql/Instruments/asianoption.hpp>
```

Include dependency graph for analytic_cont_geom_av_price.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticContinuousGeometricAveragePriceAsianEngine](#)
Pricing engine for European continuous geometric average price Asian.

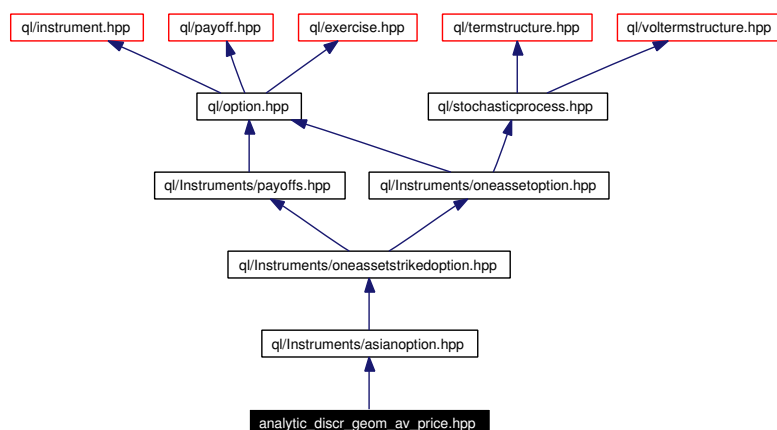
8.233 ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp File Reference

8.233.1 Detailed Description

Analytic engine for discrete geometric average price Asian.

```
#include <ql/Instruments/asianooption.hpp>
```

Include dependency graph for analytic_discr_geom_av_price.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticDiscreteGeometricAveragePriceAsianEngine](#)

Pricing engine for European discrete geometric average price Asian.

8.234 ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp File Reference

8.234.1 Detailed Description

Monte Carlo engine for discrete arithmetic average price Asian.

```
#include <ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp>
```

```
#include <ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp>
```

Include dependency graph for mc_discr_arith_av_price.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCDiscreteArithmeticAPEngine](#)
Monte Carlo pricing engine for discrete arithmetic average price Asian.

8.235 ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp File Reference

8.235.1 Detailed Description

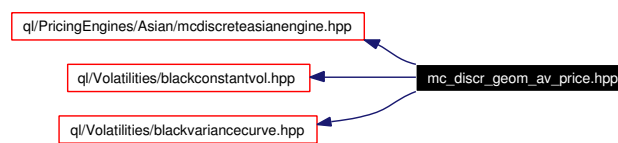
Monte Carlo engine for discrete geometric average price Asian.

```
#include <ql/PricingEngines/Asian/mcdiscreteasianengine.hpp>
```

```
#include <ql/Volatilities/blackconstantvol.hpp>
```

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for mc_discr_geom_av_price.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCDiscreteGeometricAPEngine](#)

Monte Carlo pricing engine for discrete geometric average price Asian.

8.236 ql/PricingEngines/Asian/mcdiscreteasianengine.hpp File Reference

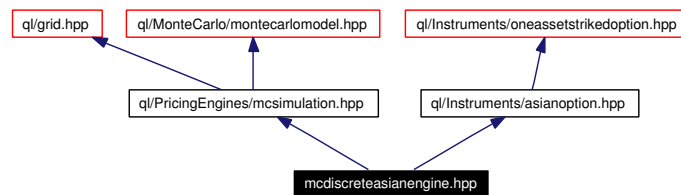
8.236.1 Detailed Description

Monte Carlo pricing engine for discrete average Asians.

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Instruments/asianoption.hpp>
```

Include dependency graph for mcdiscreteasianengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCDiscreteAveragingAsianEngine](#)
Pricing engine for discrete average Asians using Monte Carlo simulation.

8.237 ql/PricingEngines/Barrier/analyticbarrierengine.hpp File Reference

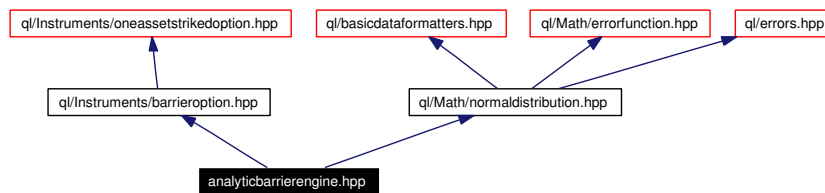
8.237.1 Detailed Description

Analytic barrier option engines.

```
#include <ql/Instruments/barrieroption.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for analyticbarrierengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticBarrierEngine](#)

Pricing engine for barrier options using analytical formulae.

8.238 ql/PricingEngines/Barrier/mcbarrierengine.hpp File Reference

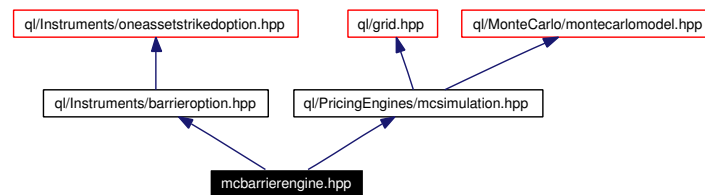
8.238.1 Detailed Description

Monte Carlo barrier option engines.

```
#include <ql/Instruments/barrieroption.hpp>
```

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

Include dependency graph for mcbarrierengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCBarrierEngine](#)

Pricing engine for barrier options using Monte Carlo simulation.

8.239 ql/PricingEngines/Basket/mcamericanbasketengine.hpp File Reference

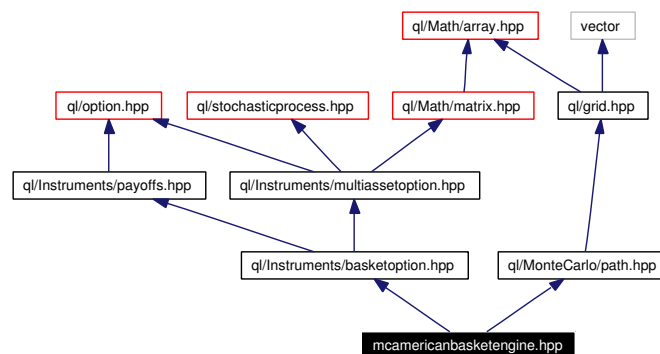
8.239.1 Detailed Description

Least-square Monte Carlo engines.

```
#include <ql/Instruments/basketoption.hpp>
```

```
#include <ql/MonteCarlo/path.hpp>
```

Include dependency graph for mcamericanbasketengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCAmericanBasketEngine](#)
least-square Monte Carlo engine

Functions

- `std::vector< Real > getAssetSequence (Real s0, const Path &path)`
- `std::vector< Real > getAntiAssetSequence (Real s0, const Path &path)`

8.240 ql/PricingEngines/Basket/mcbasketengine.hpp File Reference

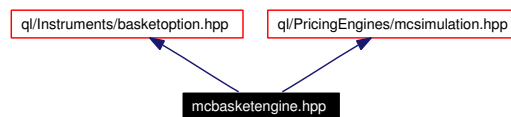
8.240.1 Detailed Description

European basket MC Engine.

```
#include <ql/Instruments/basketoption.hpp>
```

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

Include dependency graph for mcbasketengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCBasketEngine](#)
Pricing engine for basket options using Monte Carlo simulation.

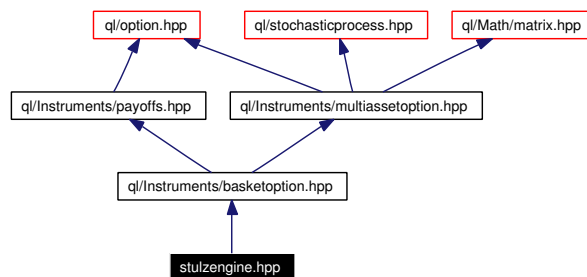
8.241 ql/PricingEngines/Basket/stulzengine.hpp File Reference

8.241.1 Detailed Description

2D European Basket formulae, due to Stulz (1982)

```
#include <ql/Instruments/basketoption.hpp>
```

Include dependency graph for stulzengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [StulzEngine](#)

Pricing engine for 2D European Baskets.

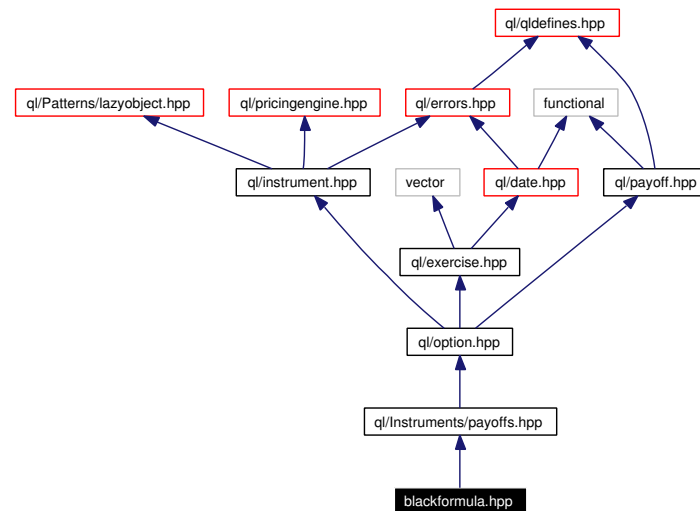
8.242 ql/PricingEngines/blackformula.hpp File Reference

8.242.1 Detailed Description

Black formula.

```
#include <ql/Instruments/payoffs.hpp>
```

Include dependency graph for blackformula.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackFormula**
Black-formula calculator.

8.243 ql/PricingEngines/blackmodel.hpp File Reference

8.243.1 Detailed Description

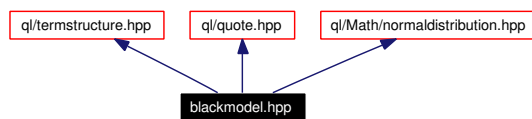
Abstract class for Black-type models (market models).

```
#include <ql/termstructure.hpp>
```

```
#include <ql/quote.hpp>
```

```
#include <ql/Math/normaldistribution.hpp>
```

Include dependency graph for blackmodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BlackModel](#)
Black-model for vanilla interest-rate derivatives.

8.244 ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp File Reference

8.244.1 Detailed Description

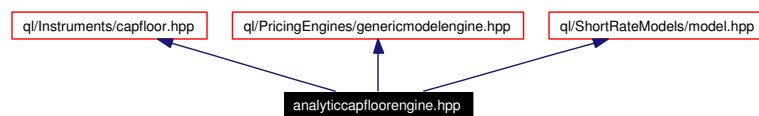
Analytic engine for caps/floors.

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

```
#include <ql/ShortRateModels/model.hpp>
```

Include dependency graph for analyticcapfloorengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticCapFloorEngine](#)
Analytic engine for cap/floor.

8.245 ql/PricingEngines/CapFloor/blackcapfloorengine.hpp File Reference

8.245.1 Detailed Description

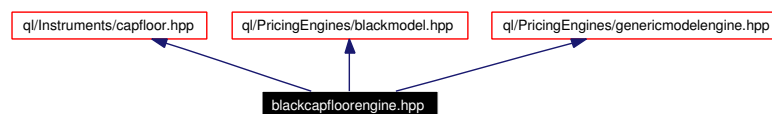
Black-formula cap/floor engine.

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/PricingEngines/blackmodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for blackcapfloorengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackCapFloorEngine**
Black-formula cap/floor engine.

8.246 ql/PricingEngines/CapFloor/discretizedcapfloor.hpp File Reference

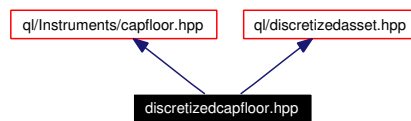
8.246.1 Detailed Description

discretized cap/floor

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for discretizedcapfloor.hpp:



Namespaces

- namespace **QuantLib**

8.247 ql/PricingEngines/CapFloor/treecapfloorengine.hpp File Reference

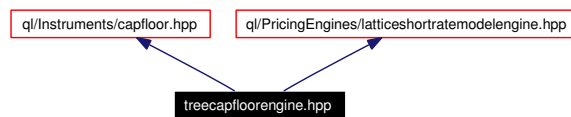
8.247.1 Detailed Description

Numerical lattice engine for cap/floors.

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

Include dependency graph for treecapfloorengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TreeCapFloorEngine](#)
Numerical lattice engine for cap/floors.

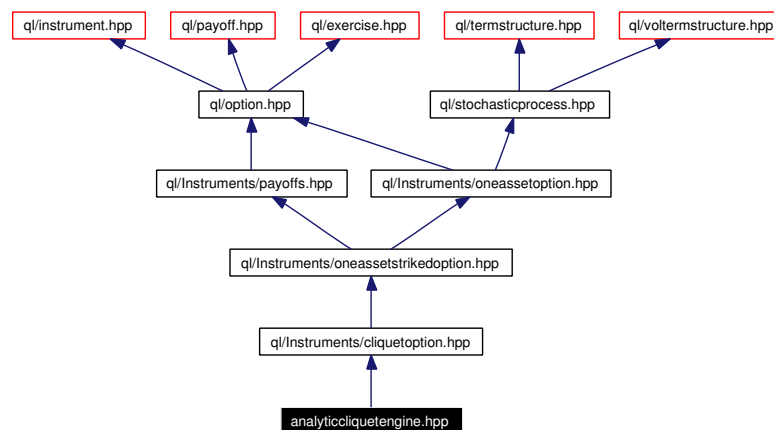
8.248 ql/PricingEngines/Cliquet/analyticcliquetengine.hpp File Reference

8.248.1 Detailed Description

Analytic Cliquet engine.

```
#include <ql/Instruments/cliquetoption.hpp>
```

Include dependency graph for analyticcliquetengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticCliquetEngine](#)
Pricing engine for Cliquet options using analytical formulae.

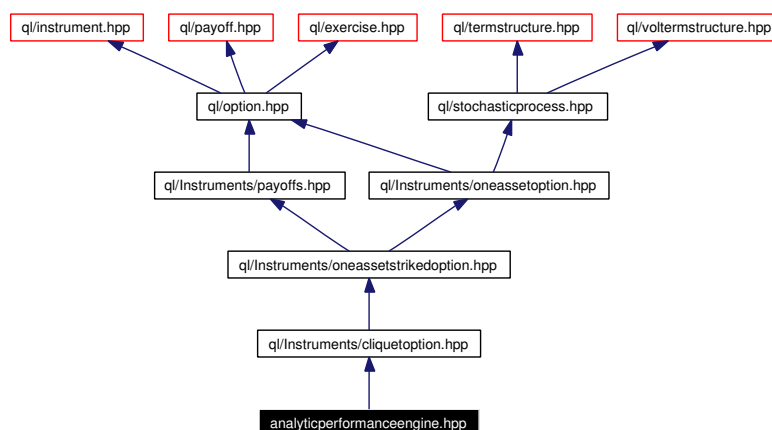
8.249 ql/PricingEngines/Cliquet/analyticperformanceengine.hpp File Reference

8.249.1 Detailed Description

Analytic performance engine.

```
#include <ql/Instruments/cliquetoption.hpp>
```

Include dependency graph for analyticperformanceengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticPerformanceEngine](#)

Pricing engine for performance options using analytical formulae.

8.250 `ql/PricingEngines/Cliquet/mccliquetengine.hpp` File Reference

8.250.1 Detailed Description

Monte Carlo Cliquet option engine.

8.251 ql/PricingEngines/Forward/forwardengine.hpp File Reference

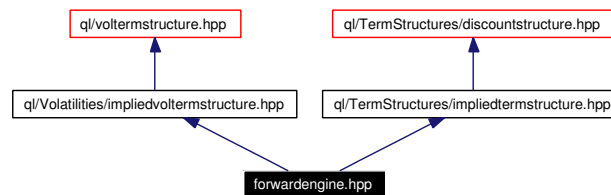
8.251.1 Detailed Description

Forward (strike-resetting) option engine.

```
#include <ql/Volatilities/impliedvoltermstructure.hpp>
```

```
#include <ql/TermStructures/impliedtermstructure.hpp>
```

Include dependency graph for forwardengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ForwardOptionArguments](#)
Arguments for forward (strike-resetting) option calculation
- class [ForwardEngine](#)
Forward engine base class.

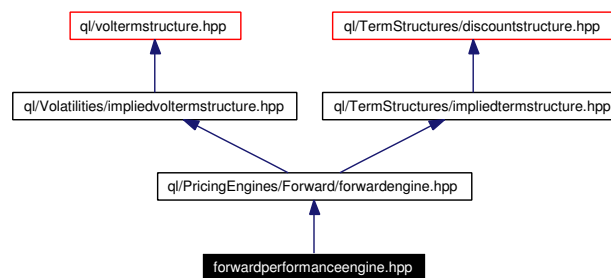
8.252 ql/PricingEngines/Forward/forwardperformanceengine.hpp File Reference

8.252.1 Detailed Description

Forward (strike-resetting) performance option engines.

```
#include <ql/PricingEngines/Forward/forwardengine.hpp>
```

Include dependency graph for forwardperformanceengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **ForwardPerformanceEngine**
Forward performance engine.

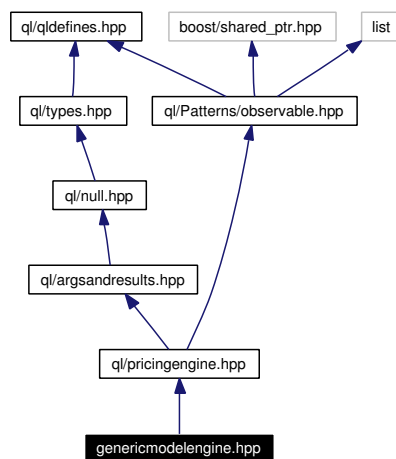
8.253 ql/PricingEngines/genericmodelengine.hpp File Reference

8.253.1 Detailed Description

Generic option engine based on a model.

```
#include <ql/pricingengine.hpp>
```

Include dependency graph for genericmodelengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [GenericModelEngine](#)
Base class for some pricing engine on a particular model.

8.254 ql/PricingEngines/latticeshortratemodelengine.hpp File Reference

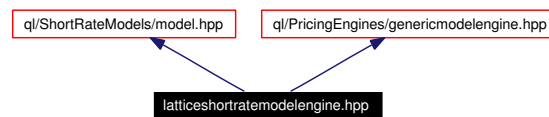
8.254.1 Detailed Description

Engine for a short-rate model specialized on a lattice.

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for latticeshortratemodelengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LatticeShortRateModelEngine](#)
Engine for a short-rate model specialized on a lattice.

8.255 ql/PricingEngines/mcsimulation.hpp File Reference

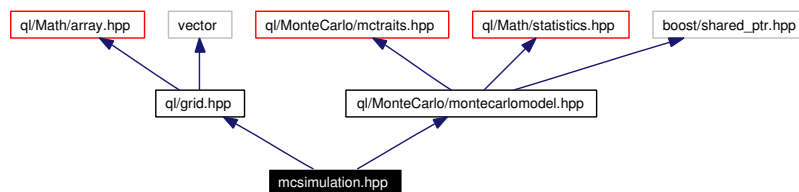
8.255.1 Detailed Description

framework for Monte Carlo engines

```
#include <ql/grid.hpp>
```

```
#include <ql/MonteCarlo/montecarlomodel.hpp>
```

Include dependency graph for mcsimulation.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [McSimulation](#)
base class for Monte Carlo engines

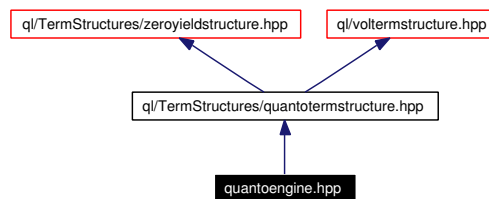
8.256 ql/PricingEngines/Quanto/quantoengine.hpp File Reference

8.256.1 Detailed Description

Quanto option engine.

```
#include <ql/TermStructures/quantotermstructure.hpp>
```

Include dependency graph for quantoengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [QuantoOptionArguments](#)
Arguments for quanto option calculation
- class [QuantoOptionResults](#)
Results from quanto option calculation
- class [QuantoEngine](#)
Quanto engine base class.

8.257 ql/PricingEngines/Swaption/blackswaptionengine.hpp File Reference

8.257.1 Detailed Description

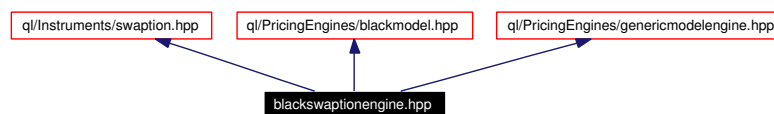
Black-formula swaption engine.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/PricingEngines/blackmodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for blackswaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackSwaptionEngine**
Black-formula swaption engine.

8.258 ql/PricingEngines/Swaption/discretizedswaption.hpp

File Reference

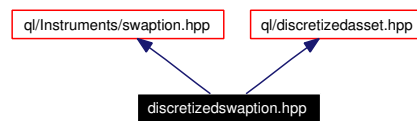
8.258.1 Detailed Description

Discretized swaption class.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/discretizedasset.hpp>
```

Include dependency graph for discretizedswaption.hpp:



Namespaces

- namespace **QuantLib**

8.259 ql/PricingEngines/Swaption/g2swaptionengine.hpp File Reference

8.259.1 Detailed Description

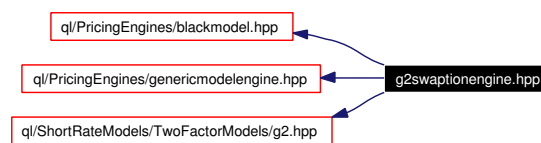
Swaption pricing engine for two-factor additive Gaussian Model G2++.

```
#include <ql/PricingEngines/blackmodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

```
#include <ql/ShortRateModels/TwoFactorModels/g2.hpp>
```

Include dependency graph for g2swaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [G2SwaptionEngine](#)
Swaption priced by means of the Black formula

8.260 ql/PricingEngines/Swaption/jamshidianswaptionengine.hpp File Reference

8.260.1 Detailed Description

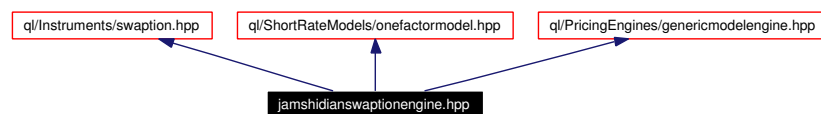
Swaption engine using Jamshidian's decomposition.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

```
#include <ql/PricingEngines/genericmodelengine.hpp>
```

Include dependency graph for jamshidianswaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [JamshidianSwaptionEngine](#)
Jamshidian swaption engine.

8.261 ql/PricingEngines/SwapOption/treeswaptionengine.hpp File Reference

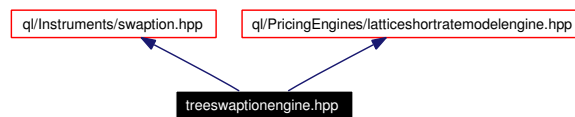
8.261.1 Detailed Description

Numerical lattice engine for swaptions.

```
#include <ql/Instruments/swaption.hpp>
```

```
#include <ql/PricingEngines/latticeshortratemodelengine.hpp>
```

Include dependency graph for treeswaptionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TreeSwaptionEngine](#)
Numerical lattice engine for swaptions.

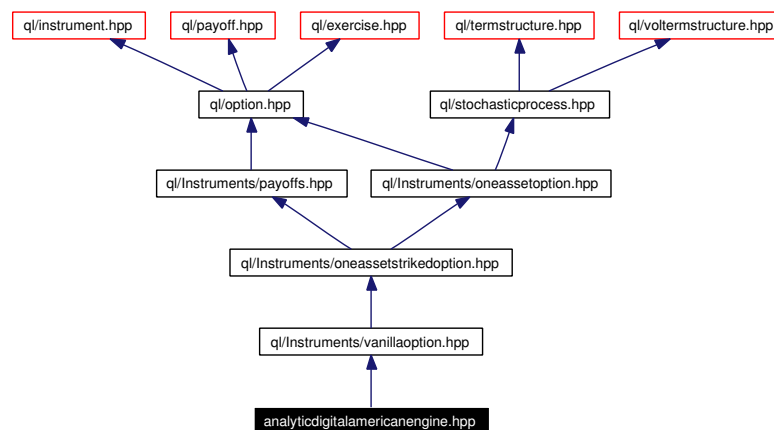
8.262 ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp File Reference

8.262.1 Detailed Description

analytic digital American option engine

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for analyticdigitalamericanengine.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `AnalyticDigitalAmericanEngine`

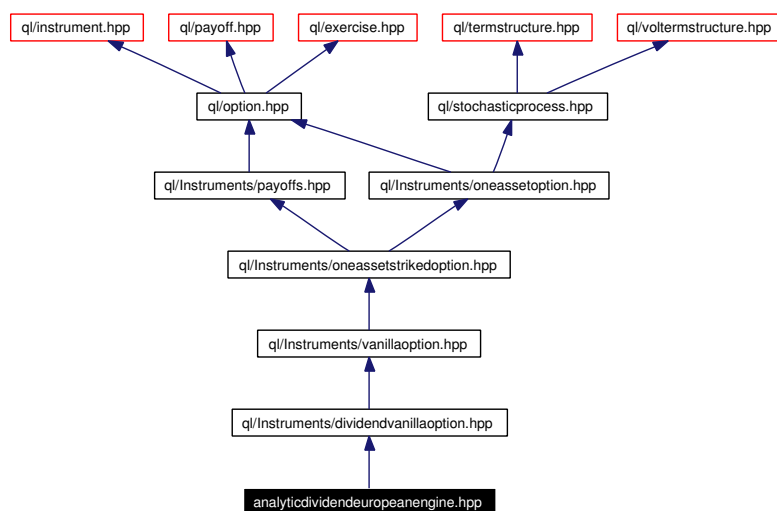
8.263 ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp File Reference

8.263.1 Detailed Description

Analytic discrete-dividend European engine.

```
#include <ql/Instruments/dividendvanillaoption.hpp>
```

Include dependency graph for analyticdividendeuropeanengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticDividendEuropeanEngine](#)
Analytic pricing engine for European options with discrete dividends.

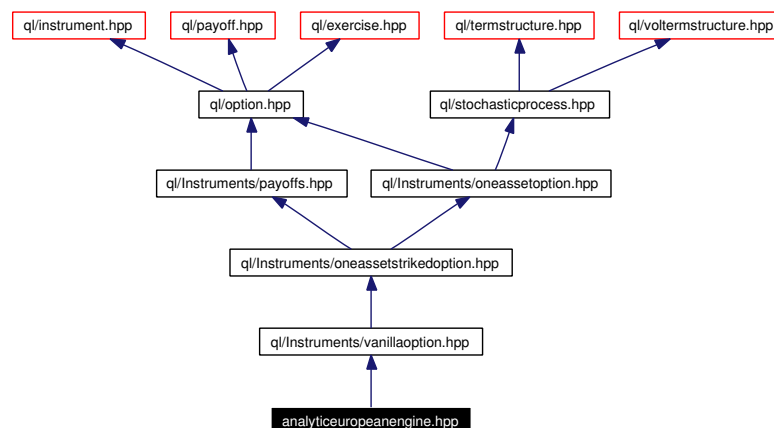
8.264 ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp File Reference

8.264.1 Detailed Description

Analytic European engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for `analyticeuropeanengine.hpp`:



Namespaces

- namespace **QuantLib**

Classes

- class [AnalyticEuropeanEngine](#)
Pricing engine for European vanilla options using analytical formulae.

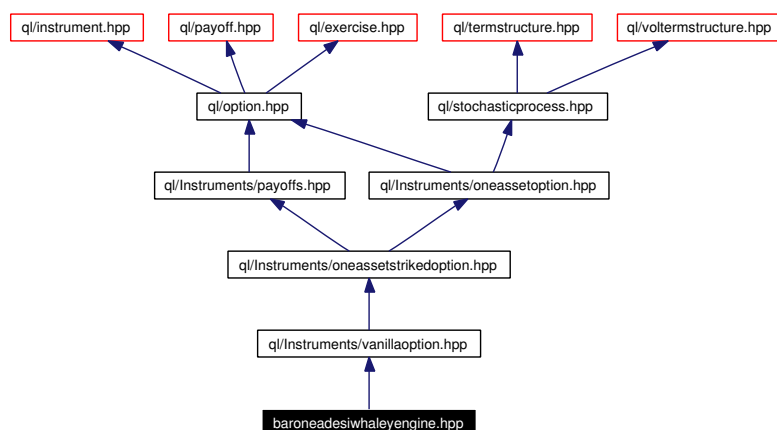
8.265 ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp File Reference

8.265.1 Detailed Description

Barone-Adesi and Whaley approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for baroneadesiwhaleyengine.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `BaroneAdesiWhaleyApproximationEngine`

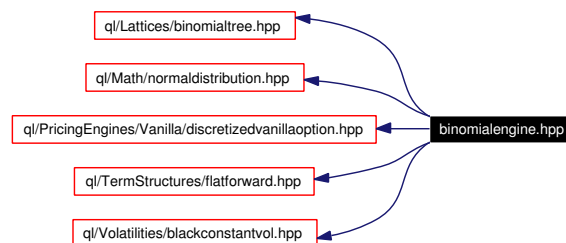
8.266 ql/PricingEngines/Vanilla/binomialengine.hpp File Reference

8.266.1 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**

Classes

- class [BinomialVanillaEngine](#)
Pricing engine for vanilla options using binomial trees.

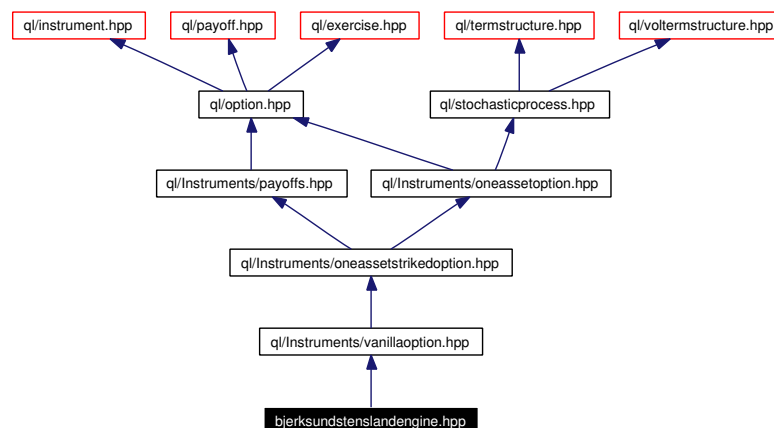
8.267 ql/PricingEngines/Vanilla/bjersundstenslandengine.hpp File Reference

8.267.1 Detailed Description

Bjersund and Stensland approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for bjersundstenslandengine.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `BjersundStenslandApproximationEngine`

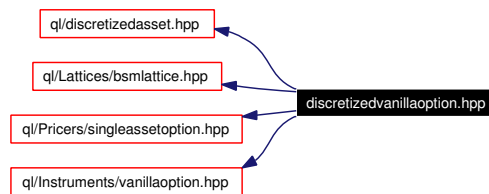
8.268 ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp File Reference

8.268.1 Detailed Description

discretized vanilla option

```
#include <ql/discretizedasset.hpp>
#include <ql/Lattices/bsmlattice.hpp>
#include <ql/Pricers/singleassetoption.hpp>
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for discretizedvanillaoption.hpp:



Namespaces

- namespace **QuantLib**

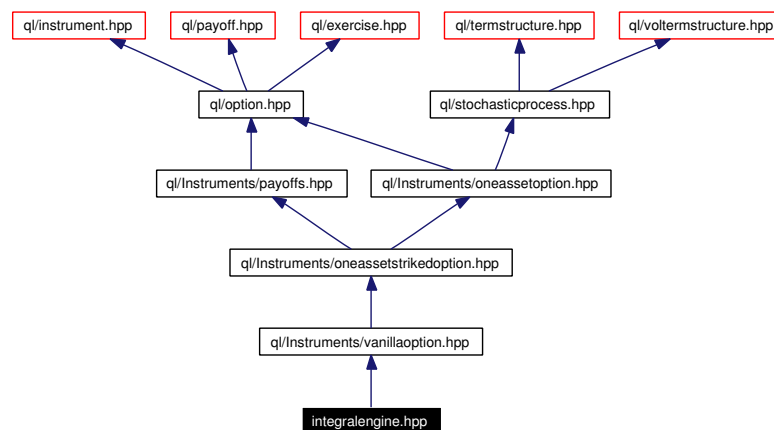
8.269 ql/PricingEngines/Vanilla/integralengine.hpp File Reference

8.269.1 Detailed Description

Integral option engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for integralengine.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `IntegralEngine`

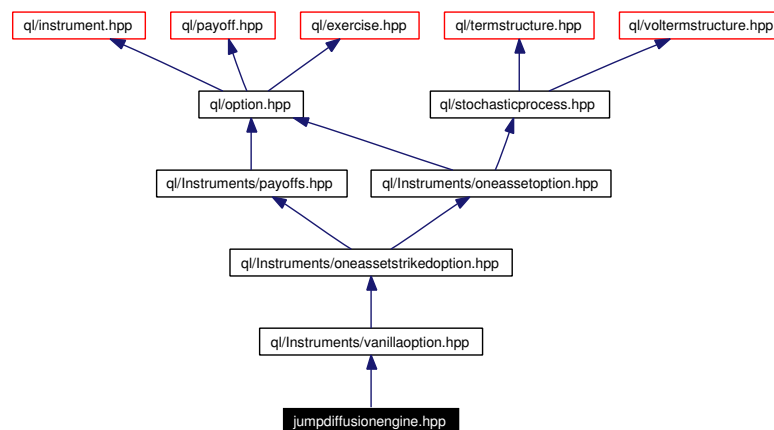
8.270 ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp File Reference

8.270.1 Detailed Description

Jump diffusion (Merton 1976) engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for jumpdiffusionengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [JumpDiffusionEngine](#)
Jump-diffusion engine for vanilla options.

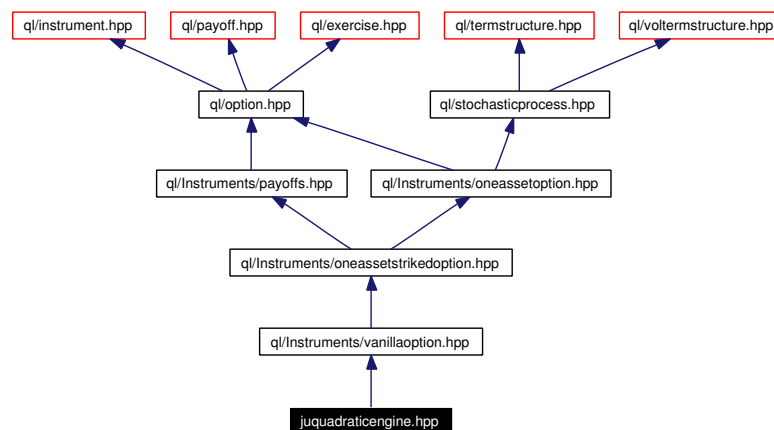
8.271 ql/PricingEngines/Vanilla/juquadraticengine.hpp File Reference

8.271.1 Detailed Description

Ju quadratic (1999) approximation engine.

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for juquadraticengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [JuQuadraticApproximationEngine](#)

8.272 ql/PricingEngines/Vanilla/mcdigitalengine.hpp File Reference

8.272.1 Detailed Description

digital option Monte Carlo engine

```
#include <ql/exercise.hpp>
```

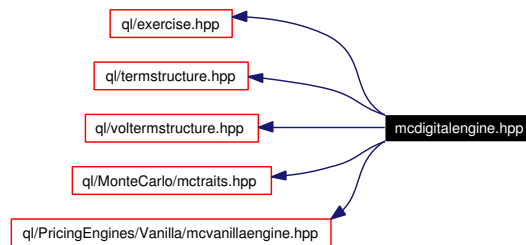
```
#include <ql/termstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/MonteCarlo/mctraits.hpp>
```

```
#include <ql/PricingEngines/Vanilla/mcvanillaengine.hpp>
```

Include dependency graph for mcdigitalengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **MCDigitalEngine**

Pricing engine for digital options using Monte Carlo simulation.

8.273 ql/PricingEngines/Vanilla/mceuropeanengine.hpp File Reference

8.273.1 Detailed Description

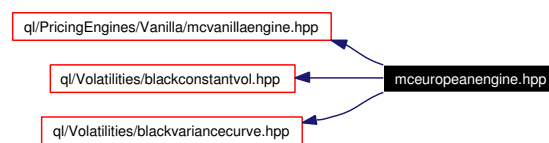
Monte Carlo European option engine.

```
#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**

Classes

- class [MCEuropeanEngine](#)

European option pricing engine using Monte Carlo simulation.

8.274 ql/PricingEngines/Vanilla/mcvanillaengine.hpp File Reference

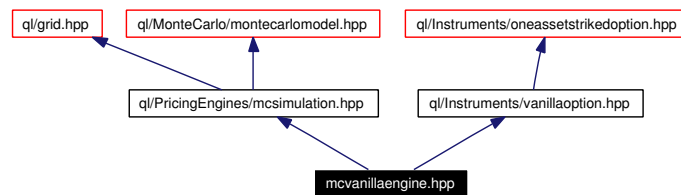
8.274.1 Detailed Description

Monte Carlo vanilla option engine.

```
#include <ql/PricingEngines/mcsimulation.hpp>
```

```
#include <ql/Instruments/vanillaoption.hpp>
```

Include dependency graph for mcvanillaengine.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MCVanillaEngine](#)

Pricing engine for vanilla options using Monte Carlo simulation.

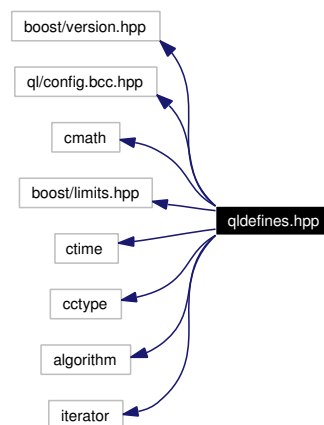
8.275 ql/qldefines.hpp File Reference

8.275.1 Detailed Description

Global definitions and compiler switches.

```
#include <boost/version.hpp>
#include <ql/config.bcc.hpp>
#include <cmath>
#include <boost/limits.hpp>
#include <ctime>
#include <cctype>
#include <algorithm>
#include <iterator>
```

Include dependency graph for qldefines.hpp:



Defines

- `#define BOOST_ENABLE_ASSERT_HANDLER`
- `#define QL_INTEGER int`
- `#define QL_BIG_INTEGER long`
- `#define QL_REAL double`
- `#define QL_VERSION "0.3.8"`
version string
- `#define QL_HEX_VERSION 0x000308f0`
version hexadecimal number
- `#define QL_LIB_VERSION "0_3_8"`
version string for output lib name
- `#define QL_DUMMY_RETURN(x)`
Is a dummy return statement required?

- `#define QL_IO_INIT`
I/O initialization.
- `#define QL_MIN_INTEGER ((std::numeric_limits<QL_INTEGER>::min)())`
- `#define QL_MAX_INTEGER ((std::numeric_limits<QL_INTEGER>::max)())`
- `#define QL_MIN_REAL -((std::numeric_limits<QL_REAL>::max)())`
- `#define QL_MAX_REAL ((std::numeric_limits<QL_REAL>::max)())`
- `#define QL_MIN_POSITIVE_REAL ((std::numeric_limits<QL_REAL>::min)())`
- `#define QL_EPSILON ((std::numeric_limits<QL_REAL>::epsilon)())`
- `#define QL_NULL_INTEGER ((std::numeric_limits<int>::max)())`
- `#define QL_NULL_REAL ((std::numeric_limits<float>::max)())`
- `#define QL_TYPENAME typename`
Blame Microsoft for this one...
- `#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`

8.276 ql/quote.hpp File Reference

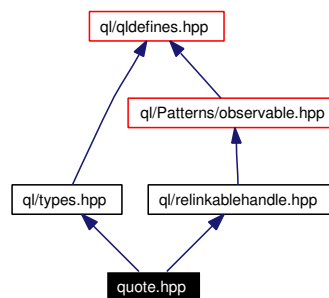
8.276.1 Detailed Description

purely virtual base class for market observables

```
#include <ql/types.hpp>
```

```
#include <ql/relinkablehandle.hpp>
```

Include dependency graph for quote.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Quote](#)
purely virtual base class for market observables
- class [SimpleQuote](#)
market element returning a stored value
- class [DerivedQuote](#)
market element whose value depends on another market element
- class [CompositeQuote](#)
market element whose value depends on two other market element

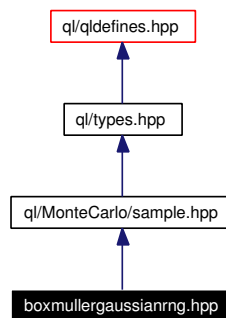
8.277 ql/RandomNumbers/boxmullergaussianrng.hpp File Reference

8.277.1 Detailed Description

Box-Muller Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for boxmullergaussianrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BoxMullerGaussianRng](#)
Gaussian random number generator.

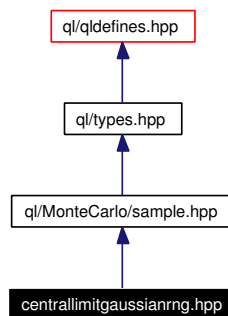
8.278 ql/RandomNumbers/centrallimitgaussianrng.hpp File Reference

8.278.1 Detailed Description

Central limit Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for centrallimitgaussianrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CLGaussianRng](#)
Gaussian random number generator.

8.279 ql/RandomNumbers/faurersg.hpp File Reference

8.279.1 Detailed Description

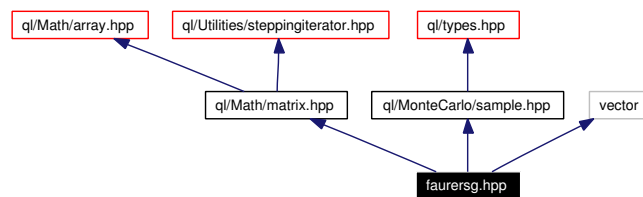
Faure low-discrepancy sequence generator.

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for faurersg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FaureRsg](#)
Faure low-discrepancy sequence generator.

8.280 ql/RandomNumbers/haltonrsg.hpp File Reference

8.280.1 Detailed Description

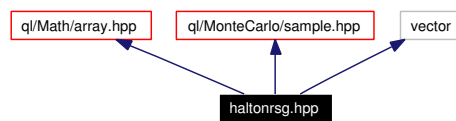
Halton low-discrepancy sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for haltonrsg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [HaltonRsg](#)
Halton low-discrepancy sequence generator.

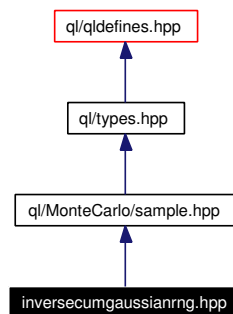
8.281 `ql/RandomNumbers/inversecumgaussianrng.hpp` File Reference

8.281.1 Detailed Description

Inverse cumulative Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for `inversecumgaussianrng.hpp`:



Namespaces

- namespace `QuantLib`

Classes

- class `ICGaussianRng`
Inverse cumulative Gaussian random number generator.

8.282 ql/RandomNumbers/inversecumgaussianrsg.hpp File Reference

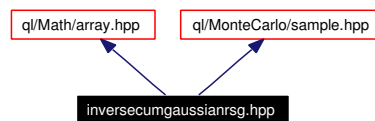
8.282.1 Detailed Description

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**

Classes

- class [ICGaussianRsg](#)
Inverse cumulative Gaussian random sequence generator.

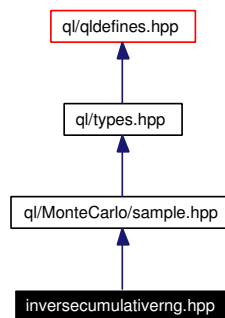
8.283 ql/RandomNumbers/inversecumulativrng.hpp File Reference

8.283.1 Detailed Description

Inverse cumulative Gaussian random-number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for inversecumulativrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InverseCumulativeRng](#)
Inverse cumulative random number generator.

8.284 ql/RandomNumbers/inversecumulativrsg.hpp File Reference

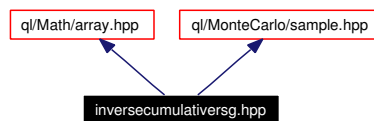
8.284.1 Detailed Description

Inverse cumulative random sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

Include dependency graph for inversecumulativrsg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [InverseCumulativeRsg](#)
Inverse cumulative random sequence generator.

8.285 ql/RandomNumbers/knuthuniformrng.hpp File Reference

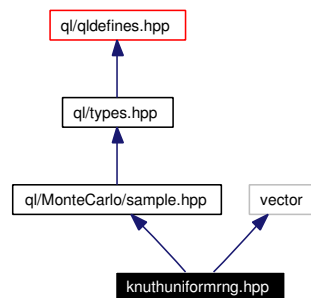
8.285.1 Detailed Description

Knuth uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for knuthuniformrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **KnuthUniformRng**
Uniform random number generator.

8.286 ql/RandomNumbers/lecuyeruniformrng.hpp File Reference

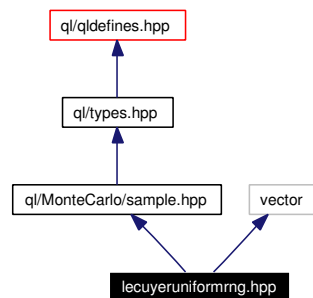
8.286.1 Detailed Description

L'Ecuyer uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for lecuyeruniformrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LecuyerUniformRng](#)
Uniform random number generator.

8.287 ql/RandomNumbers/mt19937uniformrng.hpp File Reference

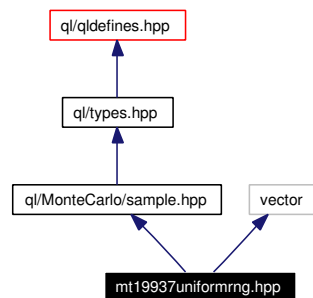
8.287.1 Detailed Description

Mersenne Twister uniform random number generator.

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for mt19937uniformrng.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [MersenneTwisterUniformRng](#)
Uniform random number generator.

8.288 ql/RandomNumbers/randomizedlds.hpp File Reference

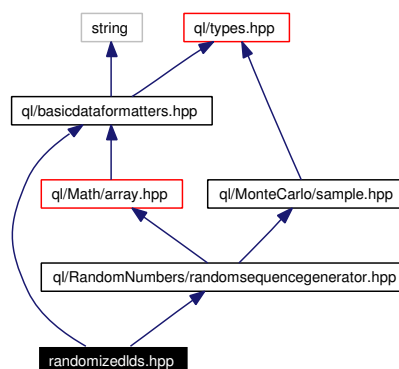
8.288.1 Detailed Description

Randomized low-discrepancy sequence.

```
#include <ql/basicdataformatters.hpp>
```

```
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
```

Include dependency graph for randomizedlds.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **RamdomizedLDS**
Randomized (random shift) low-discrepancy sequence.

8.289 ql/RandomNumbers/randomsequencegenerator.hpp File Reference

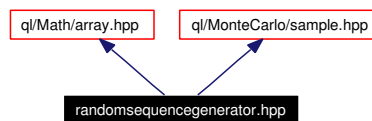
8.289.1 Detailed Description

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**

Classes

- class [RandomSequenceGenerator](#)

Random sequence generator based on a pseudo-random number generator.

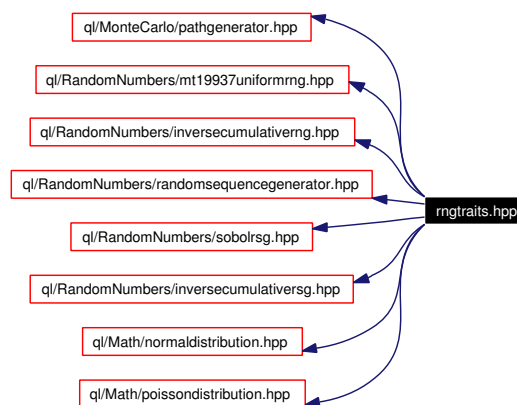
8.290 ql/RandomNumbers/rngtraits.hpp File Reference

8.290.1 Detailed Description

random-number generation policies

```
#include <ql/MonteCarlo/pathgenerator.hpp>
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
#include <ql/RandomNumbers/inversecumulativrng.hpp>
#include <ql/RandomNumbers/randomsequencegenerator.hpp>
#include <ql/RandomNumbers/sobolrsg.hpp>
#include <ql/RandomNumbers/inversecumulativsg.hpp>
#include <ql/Math/normaldistribution.hpp>
#include <ql/Math/poissondistribution.hpp>
```

Include dependency graph for rngtraits.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef GenericPseudoRandom< MersenneTwisterUniformRng, InverseCumulativeNormal > [PseudoRandom](#)
default traits for pseudo-random number generation
- typedef GenericPseudoRandom< MersenneTwisterUniformRng, InverseCumulativePoisson > [PoissonPseudoRandom](#)
traits for Poisson-distributed pseudo-random number generation
- typedef GenericLowDiscrepancy< SobolRsg, InverseCumulativeNormal > [Low-Discrepancy](#)
default traits for low-discrepancy sequence generation

8.290.2 Typedef Documentation

8.290.2.1 `typedef GenericPseudoRandom<MersenneTwisterUniformRng, InverseCumulativeNormal> PseudoRandom`

default traits for pseudo-random number generation

Tests

a sequence generator is generated and tested by comparing samples against known good values.

8.290.2.2 `typedef GenericPseudoRandom<MersenneTwisterUniformRng, InverseCumulativePoisson> PoissonPseudoRandom`

traits for Poisson-distributed pseudo-random number generation

Tests

sequence generators are generated and tested by comparing samples against known good values.

8.291 ql/RandomNumbers/seedgenerator.hpp File Reference

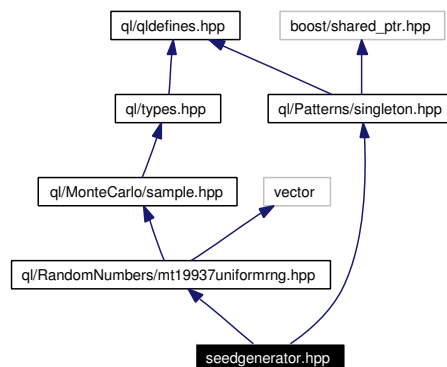
8.291.1 Detailed Description

Random seed generator.

```
#include <ql/RandomNumbers/mt19937uniformrng.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

Include dependency graph for seedgenerator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SeedGenerator](#)
Random seed generator.

8.292 ql/RandomNumbers/sobolrsg.hpp File Reference

8.292.1 Detailed Description

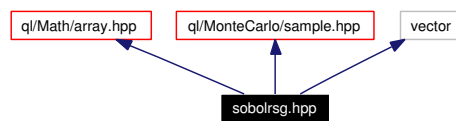
Sobol low-discrepancy sequence generator.

```
#include <ql/Math/array.hpp>
```

```
#include <ql/MonteCarlo/sample.hpp>
```

```
#include <vector>
```

Include dependency graph for sobolrsg.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SobolRsg](#)
Sobol low-discrepancy sequence generator.

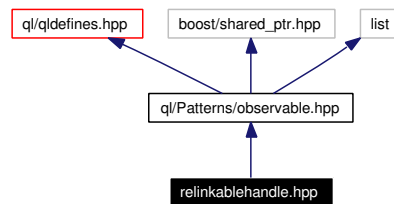
8.293 ql/relinkablehandle.hpp File Reference

8.293.1 Detailed Description

Globally accessible relinkable pointer.

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for relinkablehandle.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Link](#)
Relinkable access to a shared pointer.
- class [Handle](#)
Globally accessible relinkable pointer.

Defines

- #define [RelinkableHandle](#) `Handle`

8.293.2 Define Documentation

8.293.2.1 #define RelinkableHandle Handle

Deprecated

renamed to `Handle`

8.294 ql/schedule.hpp File Reference

8.294.1 Detailed Description

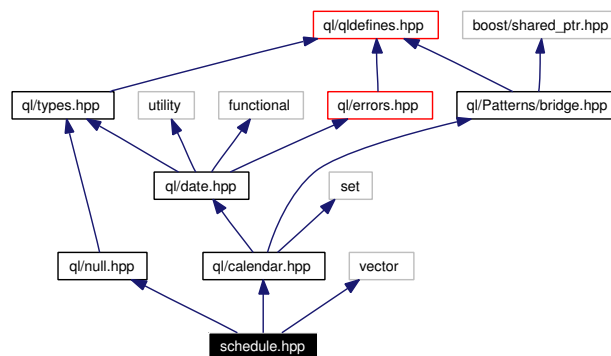
date schedule

```
#include <ql/calendar.hpp>
```

```
#include <ql/null.hpp>
```

```
#include <vector>
```

Include dependency graph for schedule.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Schedule](#)
Payment schedule.
- class [MakeSchedule](#)
helper class

8.295 ql/settings.hpp File Reference

8.295.1 Detailed Description

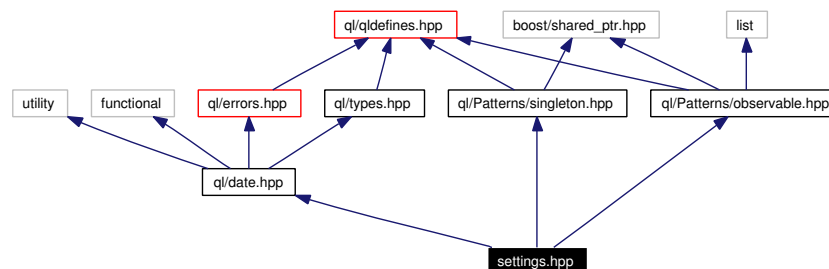
global repository for run-time library settings

```
#include <ql/date.hpp>
```

```
#include <ql/Patterns/singleton.hpp>
```

```
#include <ql/Patterns/observable.hpp>
```

Include dependency graph for settings.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Settings](#)
global repository for run-time library settings

8.296 ql/ShortRateModels/calibrationhelper.hpp File Reference

8.296.1 Detailed Description

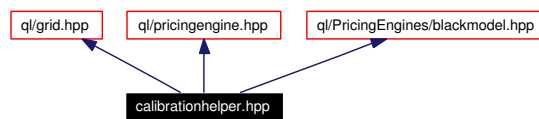
Calibration helper class.

```
#include <ql/grid.hpp>
```

```
#include <ql/pricingengine.hpp>
```

```
#include <ql/PricingEngines/blackmodel.hpp>
```

Include dependency graph for calibrationhelper.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CalibrationHelper](#)
liquid market instrument used during calibration

8.297 ql/ShortRateModels/CalibrationHelpers/caphelper.hpp File Reference

8.297.1 Detailed Description

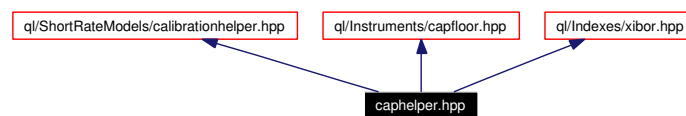
CapHelper calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/capfloor.hpp>
```

```
#include <ql/Indexes/xibor.hpp>
```

Include dependency graph for caphelper.hpp:



Namespaces

- namespace **QuantLib**

8.298 ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp File Reference

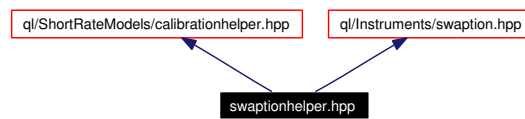
8.298.1 Detailed Description

Swaption calibration helper.

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Instruments/swaption.hpp>
```

Include dependency graph for swaptionhelper.hpp:



Namespaces

- namespace **QuantLib**

8.299 ql/ShortRateModels/model.hpp File Reference

8.299.1 Detailed Description

Abstract interest rate model class.

```
#include <ql/option.hpp>
```

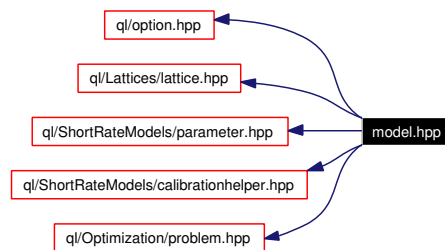
```
#include <ql/Lattices/lattice.hpp>
```

```
#include <ql/ShortRateModels/parameter.hpp>
```

```
#include <ql/ShortRateModels/calibrationhelper.hpp>
```

```
#include <ql/Optimization/problem.hpp>
```

Include dependency graph for model.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AffineModel](#)
Affine model class.
- class [TermStructureConsistentModel](#)
Term-structure consistent model class.
- class [ShortRateModel](#)
Abstract short-rate model class.

8.300 ql/ShortRateModels/onefactormodel.hpp File Reference

8.300.1 Detailed Description

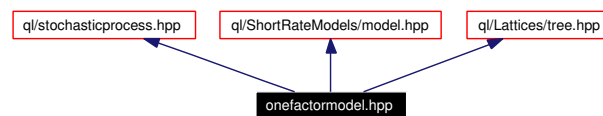
Abstract one-factor interest rate model class.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/Lattices/tree.hpp>
```

Include dependency graph for onefactormodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [OneFactorModel](#)
Single-factor short-rate model abstract class.
- class [OneFactorModel::ShortRateDynamics](#)
Base class describing the short-rate dynamics.
- class [OneFactorModel::ShortRateTree](#)
Recombining trinomial tree discretizing the state variable.
- class [OneFactorAffineModel](#)
Single-factor affine base class.

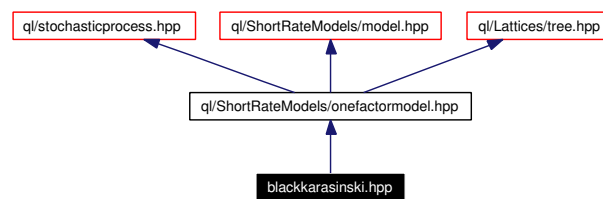
8.301 ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp File Reference

8.301.1 Detailed Description

Black-Karasinski model.

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for blackkarasinski.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BlackKarasinski](#)
Standard Black-Karasinski model class.
- class [BlackKarasinski::Dynamics](#)
Short-rate dynamics in the Black-Karasinski model.

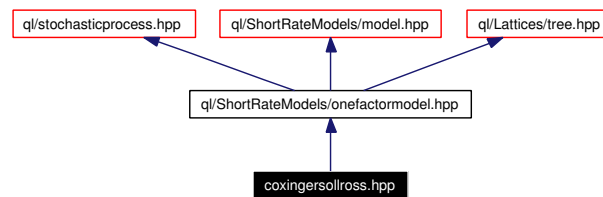
8.302 ql/ShortRateModels/OneFactorModels/coxingersollross.hpp File Reference

8.302.1 Detailed Description

Cox-Ingersoll-Ross model.

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for coxingersollross.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CoxIngersollRoss](#)
Cox-Ingersoll-Ross model class.
- class [CoxIngersollRoss::Dynamics](#)
Dynamics of the short-rate under the Cox-Ingersoll-Ross model

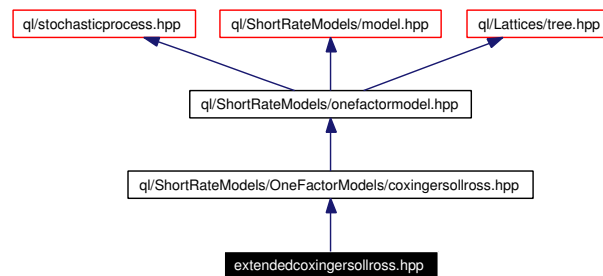
8.303 ql/ShortRateModels/OneFactorModels/extendedcoxingersollross.hpp File Reference

8.303.1 Detailed Description

Extended Cox-Ingersoll-Ross model.

```
#include <ql/ShortRateModels/OneFactorModels/coxingersollross.hpp>
```

Include dependency graph for extendedcoxingersollross.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ExtendedCoxIngersollRoss](#)
Extended Cox-Ingersoll-Ross model class.
- class [ExtendedCoxIngersollRoss::Dynamics](#)
Short-rate dynamics in the extended Cox-Ingersoll-Ross model.
- class [ExtendedCoxIngersollRoss::FittingParameter](#)
Analytical term-structure fitting parameter $\varphi(t)$.

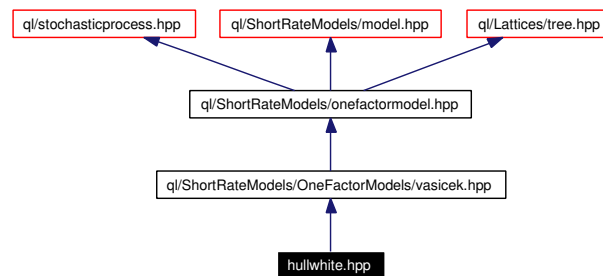
8.304 ql/ShortRateModels/OneFactorModels/hullwhite.hpp File Reference

8.304.1 Detailed Description

Hull & White (HW) model.

```
#include <ql/ShortRateModels/OneFactorModels/vasicek.hpp>
```

Include dependency graph for hullwhite.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [HullWhite](#)
Single-factor Hull-White (extended Vasicek) model class.
- class [HullWhite::Dynamics](#)
Short-rate dynamics in the Hull-White model.
- class [HullWhite::FittingParameter](#)
Analytical term-structure fitting parameter $\varphi(t)$.

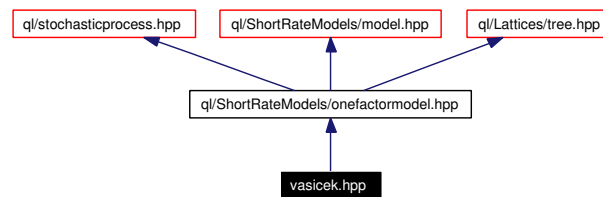
8.305 ql/ShortRateModels/OneFactorModels/vasicek.hpp File Reference

8.305.1 Detailed Description

Vasicek model class.

```
#include <ql/ShortRateModels/onefactormodel.hpp>
```

Include dependency graph for vasicek.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Vasicek](#)
Vasicek model class
- class [Vasicek::Dynamics](#)
Short-rate dynamics in the Vasicek model.

8.306 ql/ShortRateModels/parameter.hpp File Reference

8.306.1 Detailed Description

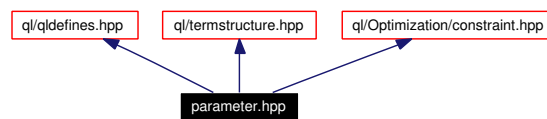
Model parameter classes.

```
#include <ql/qldefines.hpp>
```

```
#include <ql/termstructure.hpp>
```

```
#include <ql/Optimization/constraint.hpp>
```

Include dependency graph for parameter.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ParameterImpl](#)
Base class for model parameter implementation.
- class [Parameter](#)
Base class for model arguments.
- class [ConstantParameter](#)
Standard constant parameter $a(t) = a$.
- class [NullParameter](#)
Parameter which is always zero $a(t) = 0$
- class [PiecewiseConstantParameter](#)
Piecewise-constant parameter.
- class [TermStructureFittingParameter](#)
Deterministic time-dependent parameter used for yield-curve fitting.

8.307 ql/ShortRateModels/twofactormodel.hpp File Reference

8.307.1 Detailed Description

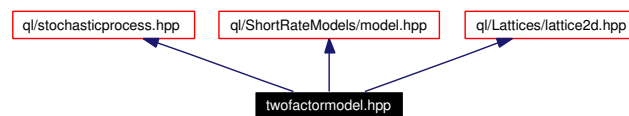
Abstract two-factor interest rate model class.

```
#include <ql/stochasticprocess.hpp>
```

```
#include <ql/ShortRateModels/model.hpp>
```

```
#include <ql/Lattices/lattice2d.hpp>
```

Include dependency graph for twofactormodel.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [TwoFactorModel](#)
Abstract base-class for two-factor models.
- class [TwoFactorModel::ShortRateDynamics](#)
Class describing the dynamics of the two state variables.
- class [TwoFactorModel::ShortRateTree](#)
Recombining two-dimensional tree discretizing the state variable.

8.308 ql/ShortRateModels/TwoFactorModels/g2.hpp File Reference

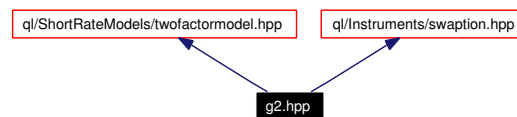
8.308.1 Detailed Description

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**

Classes

- class [G2](#)
Two-additive-factor gaussian model class.
- class [G2::FittingParameter](#)
Analytical term-structure fitting parameter $\varphi(t)$.

8.309 ql/solver1d.hpp File Reference

8.309.1 Detailed Description

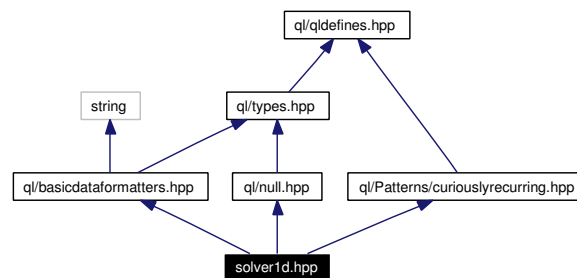
Abstract 1-D solver class.

```
#include <ql/null.hpp>
```

```
#include <ql/basicdataformatters.hpp>
```

```
#include <ql/Patterns/curiouslyrecurring.hpp>
```

Include dependency graph for solver1d.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Solver1D](#)
Base class for 1-D solvers.

Defines

- `#define MAX_FUNCTION_EVALUATIONS 100`

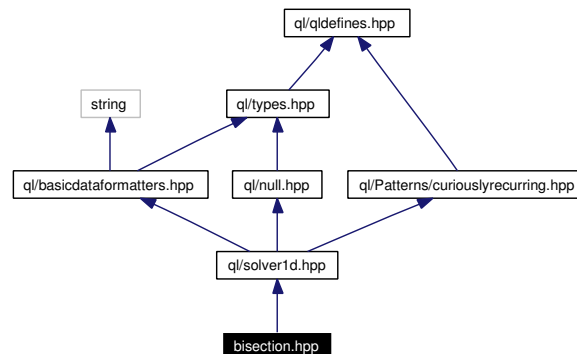
8.310 ql/Solvers1D/bisection.hpp File Reference

8.310.1 Detailed Description

bisection 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for bisection.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Bisection](#)
Bisection 1-D solver

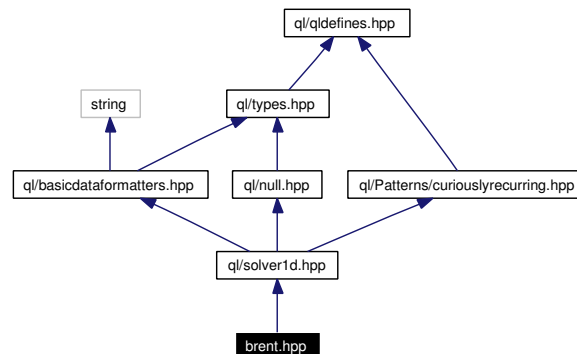
8.311 ql/Solvers1D/brent.hpp File Reference

8.311.1 Detailed Description

Brent 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for brent.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Brent](#)
Brent 1-D solver

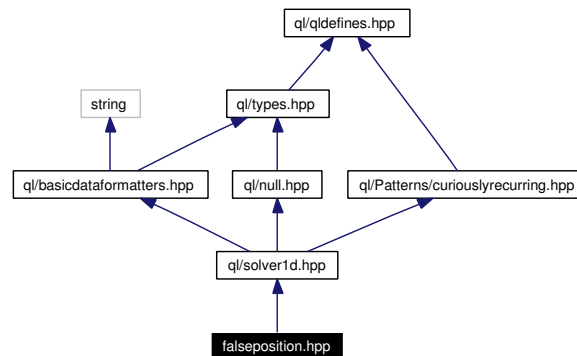
8.312 ql/Solvers1D/falseposition.hpp File Reference

8.312.1 Detailed Description

false-position 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for falseposition.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FalsePosition](#)
False position 1-D solver.

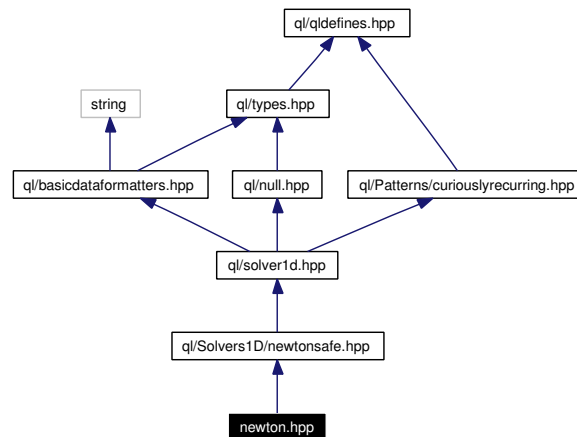
8.313 ql/Solvers1D/newton.hpp File Reference

8.313.1 Detailed Description

Newton 1-D solver.

```
#include <ql/Solvers1D/newtonsafe.hpp>
```

Include dependency graph for newton.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Newton](#)
Newton 1-D solver

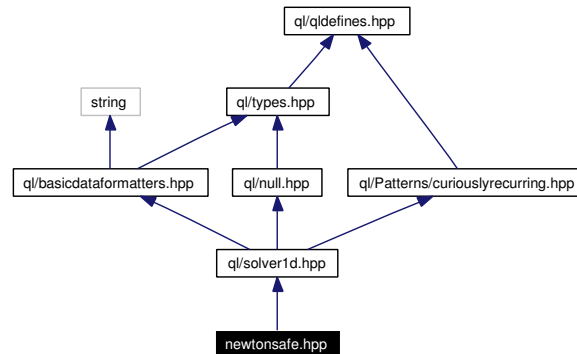
8.314 ql/Solvers1D/newtonsafe.hpp File Reference

8.314.1 Detailed Description

Safe (bracketed) Newton 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for newtonsafe.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [NewtonSafe](#)
safe Newton 1-D solver

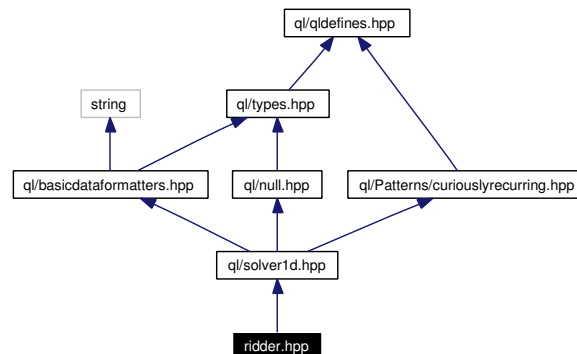
8.315 ql/Solvers1D/ridder.hpp File Reference

8.315.1 Detailed Description

Ridder 1-D solver.

```
#include <ql/solver1d.hpp>
```

Include dependency graph for ridder.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Ridder](#)
Ridder 1-D solver

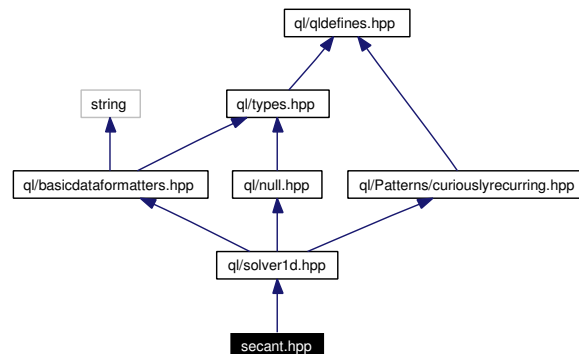
8.316 ql/Solvers1D/secant.hpp File Reference

8.316.1 Detailed Description

secant 1-D solver

```
#include <ql/solver1d.hpp>
```

Include dependency graph for secant.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [Secant](#)
Secant 1-D solver

8.317 ql/stochasticprocess.hpp File Reference

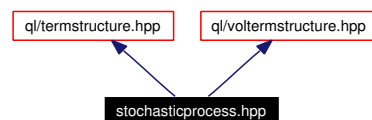
8.317.1 Detailed Description

stochastic processes

```
#include <ql/termstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for stochasticprocess.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [StochasticProcess](#)
Stochastic process class.
- class [StochasticProcess::discretization](#)
discretization of a stochastic process over a given time interval
- class [EulerDiscretization](#)
Euler discretization for stochastic processes.
- class [GeometricBrownianMotionProcess](#)
Geometric brownian motion process.
- class [BlackScholesProcess](#)
Black-Scholes stochastic process.
- class [Merton76Process](#)
Merton-76 jump-diffusion process.
- class [OrnsteinUhlenbeckProcess](#)
Ornstein-Uhlenbeck process class.
- class [SquareRootProcess](#)
Square-root process class.

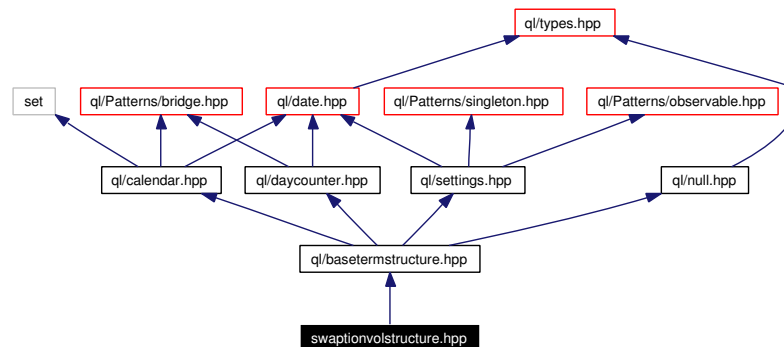
8.318 ql/swaptionvolstructure.hpp File Reference

8.318.1 Detailed Description

Swaption volatility structure.

```
#include <ql/basetermstructure.hpp>
```

Include dependency graph for swaptionvolstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **SwaptionVolatilityStructure**
Swaption-volatility structure

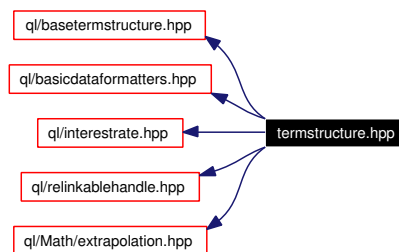
8.319 ql/termstructure.hpp File Reference

8.319.1 Detailed Description

Term structure.

```
#include <ql/basetermstructure.hpp>
#include <ql/basicdataformatters.hpp>
#include <ql/interestrates.hpp>
#include <ql/relinkablehandle.hpp>
#include <ql/Math/extrapolation.hpp>
```

Include dependency graph for termstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [YieldTermStructure](#)
Interest-rate term structure.

Typedefs

- typedef YieldTermStructure [TermStructure](#)

8.319.2 Typedef Documentation

8.319.2.1 typedef YieldTermStructure TermStructure

Deprecated

renamed to [YieldTermStructure](#)

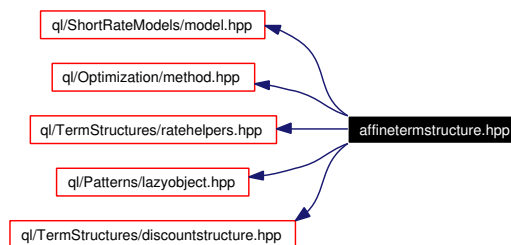
8.320 ql/TermStructures/affinetermstructure.hpp File Reference

8.320.1 Detailed Description

Affine term structure.

```
#include <ql/ShortRateModels/model.hpp>
#include <ql/Optimization/method.hpp>
#include <ql/TermStructures/ratehelpers.hpp>
#include <ql/Patterns/lazyobject.hpp>
#include <ql/TermStructures/discountstructure.hpp>
```

Include dependency graph for affinetermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [AffineTermStructure](#)
Term-structure implied by an affine model.

8.321 ql/TermStructures/compoundforward.hpp File Reference

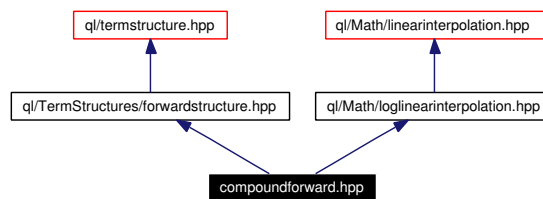
8.321.1 Detailed Description

compounded forward term structure

```
#include <ql/TermStructures/forwardstructure.hpp>
```

```
#include <ql/Math/loglinearinterpolation.hpp>
```

Include dependency graph for compoundforward.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CompoundForward](#)
compound-forward structure

8.322 ql/TermStructures/discountcurve.hpp File Reference

8.322.1 Detailed Description

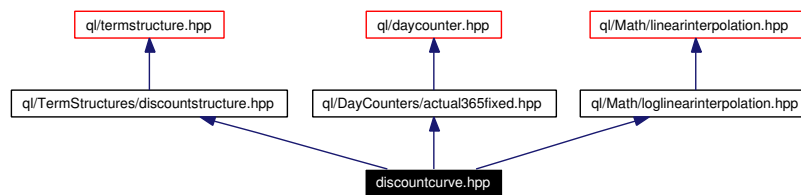
pre-bootstrapped discount factor structure

```
#include <ql/TermStructures/discountstructure.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

```
#include <ql/Math/loglinearinterpolation.hpp>
```

Include dependency graph for discountcurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DiscountCurve](#)

Term structure based on loglinear interpolation of discount factors.

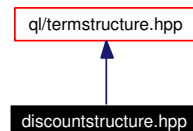
8.323 ql/TermStructures/discountstructure.hpp File Reference

8.323.1 Detailed Description

Discount-based yield term structure.

```
#include <ql/termstructure.hpp>
```

Include dependency graph for discountstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DiscountStructure](#)
Discount factor term structure.

8.324 ql/TermStructures/driftermstructure.hpp File Reference

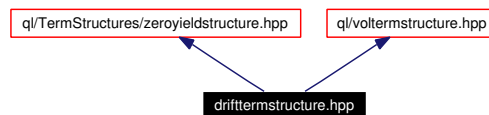
8.324.1 Detailed Description

Drift term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for driftermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [DriftTermStructure](#)
Drift term structure.

8.325 ql/TermStructures/extendeddiscountcurve.hpp File Reference

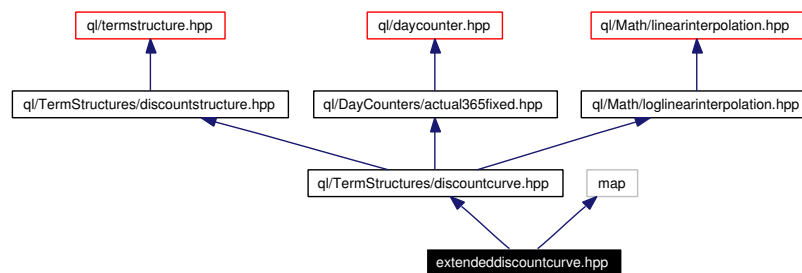
8.325.1 Detailed Description

discount factor structure with detailed compound-forward calculation

```
#include <ql/TermStructures/discountcurve.hpp>
```

```
#include <map>
```

Include dependency graph for extendeddiscountcurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ExtendedDiscountCurve](#)
Term structure based on loglinear interpolation of discount factors.

8.326 ql/TermStructures/flatforward.hpp File Reference

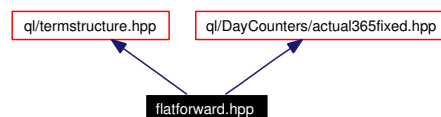
8.326.1 Detailed Description

flat forward rate term structure

```
#include <ql/termstructure.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for flatforward.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [FlatForward](#)
Flat interest-rate curve.

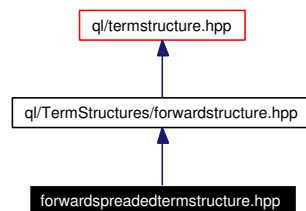
8.327 ql/TermStructures/forwardspreadedtermstructure.hpp File Reference

8.327.1 Detailed Description

Forward-spreaded term structure.

```
#include <ql/TermStructures/forwardstructure.hpp>
```

Include dependency graph for forwardspreadedtermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ForwardSpreadedTermStructure](#)
Term structure with added spread on the instantaneous forward rate.

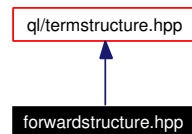
8.328 ql/TermStructures/forwardstructure.hpp File Reference

8.328.1 Detailed Description

Forward-based yield term structure.

```
#include <ql/termstructure.hpp>
```

Include dependency graph for forwardstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ForwardRateStructure](#)
Forward rate term structure.

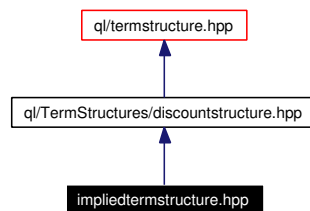
8.329 ql/TermStructures/IMPLIEDTERMSTRUCTURE.hpp File Reference

8.329.1 Detailed Description

Implied term structure.

```
#include <ql/TermStructures/discountstructure.hpp>
```

Include dependency graph for IMPLIEDTERMSTRUCTURE.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ImpliedTermStructure](#)
Implied term structure at a given date in the future.

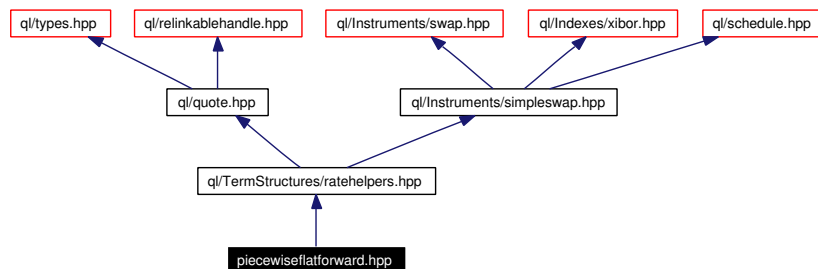
8.330 ql/TermStructures/piecewiseflatforward.hpp File Reference

8.330.1 Detailed Description

piecewise flat forward term structure

```
#include <ql/TermStructures/ratehelpers.hpp>
```

Include dependency graph for piecewiseflatforward.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [PiecewiseFlatForward](#)
Piecewise flat forward term structure.

8.331 ql/TermStructures/quantotermstructure.hpp File Reference

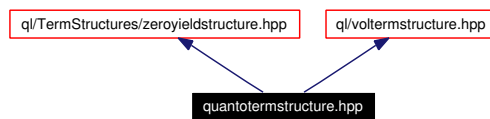
8.331.1 Detailed Description

Quanto term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for quantotermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [QuantoTermStructure](#)
Quanto term structure.

8.332 ql/TermStructures/ratehelpers.hpp File Reference

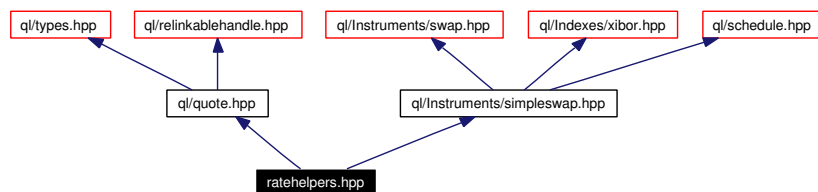
8.332.1 Detailed Description

rate helpers base class

```
#include <ql/quote.hpp>
```

```
#include <ql/Instruments/simpleswap.hpp>
```

Include dependency graph for ratehelpers.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [RateHelper](#)
Base class for rate helpers.
- class [DepositRateHelper](#)
Deposit rate.
- class [FraRateHelper](#)
Forward rate agreement.
- class [FuturesRateHelper](#)
Interest-rate futures.
- class [SwapRateHelper](#)
Swap rate

8.333 ql/TermStructures/zerocurve.hpp File Reference

8.333.1 Detailed Description

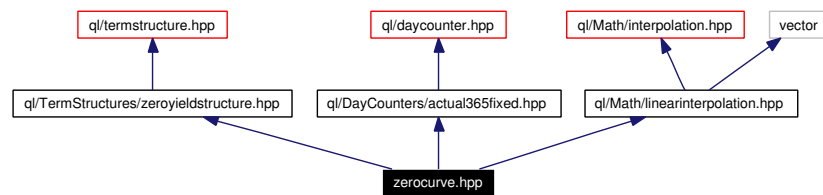
pre-bootstrapped zero curve structure

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

```
#include <ql/Math/linearinterpolation.hpp>
```

Include dependency graph for zerocurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ZeroCurve](#)

Term structure based on linear interpolation of zero yields.

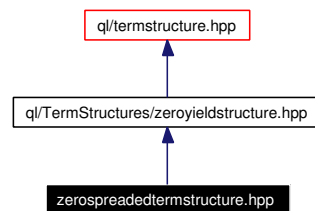
8.334 ql/TermStructures/zerospreadedtermstructure.hpp File Reference

8.334.1 Detailed Description

Zero spreaded term structure.

```
#include <ql/TermStructures/zeroyieldstructure.hpp>
```

Include dependency graph for zerospreadedtermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ZeroSpreadedTermStructure](#)
Term structure with an added spread on the zero yield rate.

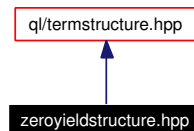
8.335 ql/TermStructures/zeroyieldstructure.hpp File Reference

8.335.1 Detailed Description

Zero-yield based term structure.

```
#include <ql/termstructure.hpp>
```

Include dependency graph for zeroyieldstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ZeroYieldStructure](#)
Zero-yield term structure.

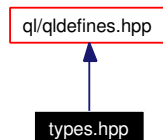
8.336 ql/types.hpp File Reference

8.336.1 Detailed Description

Custom types.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for types.hpp:



Namespaces

- namespace **QuantLib**

Typedefs

- typedef QL_INTEGER [Integer](#)
integer number
- typedef QL_BIG_INTEGER [BigInteger](#)
large integer number
- typedef unsigned QL_INTEGER [Natural](#)
positive integer
- typedef unsigned QL_BIG_INTEGER [BigNatural](#)
large positive integer
- typedef QL_REAL [Real](#)
real number
- typedef [Real](#) [Decimal](#)
decimal number
- typedef QL_SIZE_T [Size](#)
size of a container
- typedef [Real](#) [Time](#)
continuous quantity with 1-year units
- typedef [Real](#) [DiscountFactor](#)
discount factor between dates
- typedef [Real](#) [Rate](#)

interest rates

- typedef [Real Spread](#)
spreads on interest rates
- typedef [Real Volatility](#)
volatility

8.337 ql/Utilities/combiningiterator.hpp File Reference

8.337.1 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**

Classes

- class [combining_iterator](#)
Iterator mapping a function to a set of underlying sequences.

Functions

- `template<class I, class F> combining_iterator< typename 1< I >::value_type, F > make_combining_iterator (I it1, I it2, F f)`

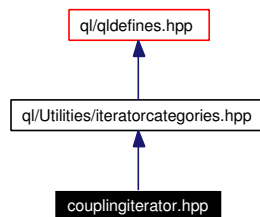
8.338 ql/Utilities/couplingiterator.hpp File Reference

8.338.1 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**

Classes

- class [coupling_iterator](#)

Iterator mapping a function to a pair of underlying sequences.

8.339 ql/Utilities/filteringiterator.hpp File Reference

8.339.1 Detailed Description

Iterator filtering undesired data.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for filteringiterator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [filtering_iterator](#)
Iterator filtering undesired data.

8.340 ql/Utilities/iteratorcategories.hpp File Reference

8.340.1 Detailed Description

Lowest common denominator between two iterator categories.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for iteratorcategories.hpp:



Namespaces

- namespace **QuantLib**

Classes

- struct [lowest_category_iterator](#)
most generic of two given iterator categories

8.341 ql/Utilities/processingiterator.hpp File Reference

8.341.1 Detailed Description

Iterator mapping a unary function to an underlying sequence.

```
#include <ql/qldefines.hpp>
```

Include dependency graph for processingiterator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [processing_iterator](#)
Iterator mapping a unary function to an underlying sequence.

Functions

- `template<class I, class F> processing_iterator< I, F > make_processing_iterator (I it, F p)`

8.342 ql/Utilities/steppingiterator.hpp File Reference

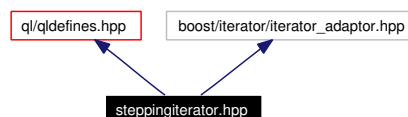
8.342.1 Detailed Description

Iterator advancing in constant steps.

```
#include <ql/qldefines.hpp>
```

```
#include <boost/iterator/iterator_adaptor.hpp>
```

Include dependency graph for steppingiterator.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [step_iterator](#)
Iterator advancing in constant steps.
- class [stepping_iterator](#)
Iterator advancing in constant steps.

8.343 ql/Volatilities/blackconstantvol.hpp File Reference

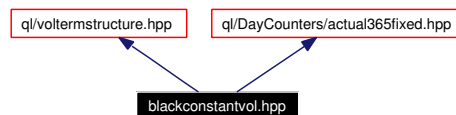
8.343.1 Detailed Description

Black constant volatility, no time dependence, no strike dependence.

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for blackconstantvol.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BlackConstantVol](#)
Constant Black volatility, no time-strike dependence.

8.344 ql/Volatilities/blackvariancecurve.hpp File Reference

8.344.1 Detailed Description

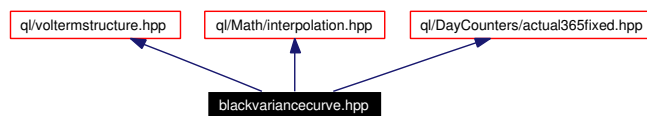
Black volatility curve modelled as variance curve.

```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/Math/interpolation.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for blackvariancecurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BlackVarianceCurve](#)
Black volatility curve modelled as variance curve.

8.345 ql/Volatilities/blackvariancesurface.hpp File Reference

8.345.1 Detailed Description

Black volatility surface modelled as variance surface.

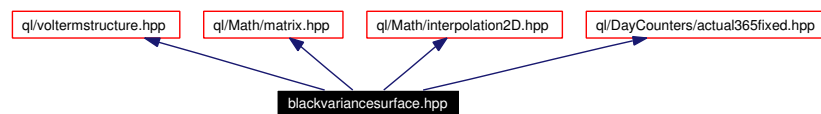
```
#include <ql/voltermstructure.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Math/interpolation2D.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for blackvariancesurface.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [BlackVarianceSurface](#)
Black volatility surface modelled as variance surface.

8.346 ql/Volatilities/capflatvolvector.hpp File Reference

8.346.1 Detailed Description

Cap/floor at-the-money flat volatility vector.

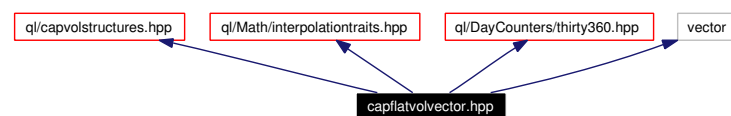
```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/Math/interpolationtraits.hpp>
```

```
#include <ql/DayCounters/thirty360.hpp>
```

```
#include <vector>
```

Include dependency graph for capflatvolvector.hpp:



Namespaces

- namespace `QuantLib`

Classes

- class `CapVolatilityVector`
Cap/floor at-the-money term-volatility vector.

Typedefs

- typedef `CapVolatilityVector` `CapFlatVolatilityVector`

8.346.2 Typedef Documentation

8.346.2.1 typedef `CapVolatilityVector` `CapFlatVolatilityVector`

Deprecated

renamed to `CapVolatilityVector`

8.347 ql/Volatilities/capletconstantvol.hpp File Reference

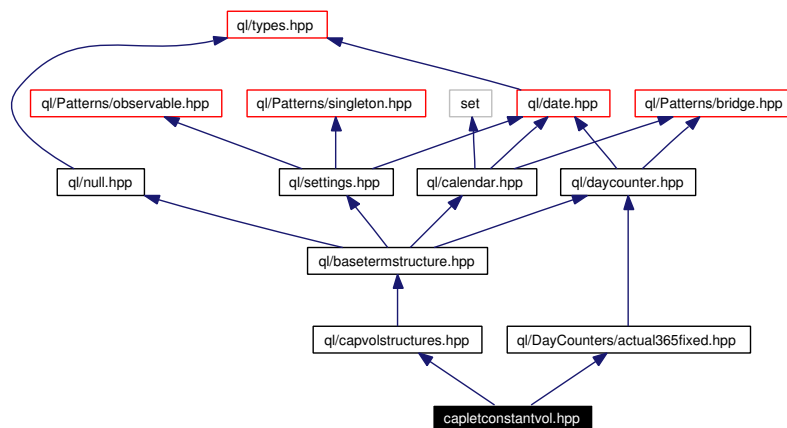
8.347.1 Detailed Description

Constant caplet volatility.

```
#include <ql/capvolstructures.hpp>
```

```
#include <ql/DayCounters/actual365fixed.hpp>
```

Include dependency graph for capletconstantvol.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [CapletConstantVolatility](#)
Constant caplet volatility, no time-strike dependence.

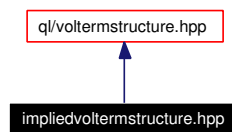
8.348 ql/Volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp File Reference

8.348.1 Detailed Description

Implied Black Vol Term Structure.

```
#include <ql/voltermstructure.hpp>
```

Include dependency graph for IMPLIEDVOLTERMSTRUCTURE.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [ImpliedVolTermStructure](#)
Implied vol term structure at a given date in the future.

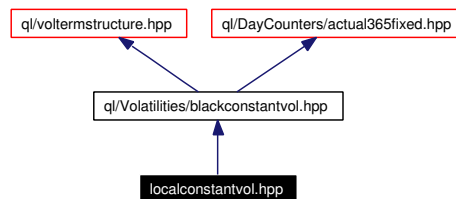
8.349 ql/Volatilities/localconstantvol.hpp File Reference

8.349.1 Detailed Description

Local constant volatility, no time dependence, no asset dependence.

```
#include <ql/Volatilities/blackconstantvol.hpp>
```

Include dependency graph for localconstantvol.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LocalConstantVol](#)
Constant local volatility, no time-strike dependence.

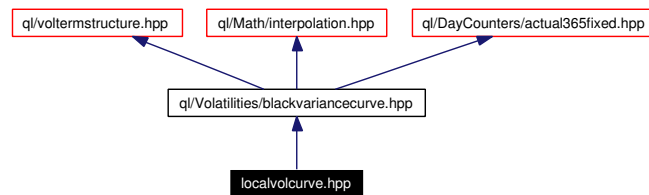
8.350 ql/Volatilities/localvolcurve.hpp File Reference

8.350.1 Detailed Description

Local volatility curve derived from a Black curve.

```
#include <ql/Volatilities/blackvariancecurve.hpp>
```

Include dependency graph for localvolcurve.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [LocalVolCurve](#)
Local volatility curve derived from a Black curve.

8.351 ql/Volatilities/localvolsurface.hpp File Reference

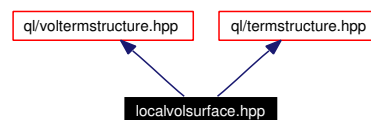
8.351.1 Detailed Description

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**

Classes

- class [LocalVolSurface](#)
Local volatility surface derived from a Black vol surface.

8.352 ql/Volatilities/swaptionvolmatrix.hpp File Reference

8.352.1 Detailed Description

Swaption at-the-money volatility matrix.

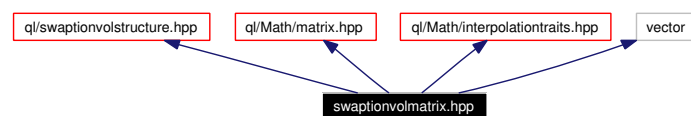
```
#include <ql/swaptionvolstructure.hpp>
```

```
#include <ql/Math/matrix.hpp>
```

```
#include <ql/Math/interpolationtraits.hpp>
```

```
#include <vector>
```

Include dependency graph for swaptionvolmatrix.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class [SwaptionVolatilityMatrix](#)
At-the-money swaption-volatility matrix.

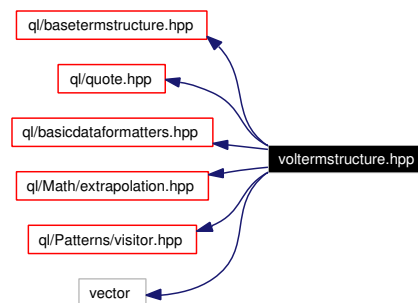
8.353 ql/voltermstructure.hpp File Reference

8.353.1 Detailed Description

Volatility term structures.

```
#include <ql/basetermstructure.hpp>
#include <ql/quote.hpp>
#include <ql/basicdataformatters.hpp>
#include <ql/Math/extrapolation.hpp>
#include <ql/Patterns/visitor.hpp>
#include <vector>
```

Include dependency graph for voltermstructure.hpp:



Namespaces

- namespace **QuantLib**

Classes

- class **BlackVolTermStructure**
Black-volatility term structure.
- class **BlackVolatilityTermStructure**
Black-volatility term structure.
- class **BlackVarianceTermStructure**
Black variance term structure.
- class **LocalVolTermStructure**
Local-volatility term structure.

Chapter 9

QuantLib Example Documentation

9.1 AmericanOption.cpp

This example calculates American options using different methods

```
#include <ql/quantlib.hpp>

using namespace QuantLib;

int main(int, char* [])
{
    try {
        QL_IO_INIT

        std::cout << "Using " << QL_VERSION << std::endl << std::endl;

        // our option
        Option::Type type(Option::Put);
        Real underlying = 36;
        Real strike = 40;
        Spread dividendYield = 0.00;
        Rate riskFreeRate = 0.06;
        Volatility volatility = 0.20;

        Date todaysDate(15, May, 1998);
        Date settlementDate(17, May, 1998);
        Settings::instance().setEvaluationDate(todaysDate);

        Date exerciseDate(17, May, 1999);
        DayCounter dayCounter = Actual365Fixed();
        Time maturity = dayCounter.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;

Real 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;

boost::shared_ptr<Exercise> exercise(
    new EuropeanExercise(exerciseDate));
boost::shared_ptr<Exercise> amExercise(
    new AmericanExercise(settlementDate,
                          exerciseDate));
boost::shared_ptr<Exercise> berExercise(new BermudanExercise(exDates));

Handle<Quote> underlyingH(
    boost::shared_ptr<Quote>(new SimpleQuote(underlying)));

// bootstrap the yield/dividend/vol curves
Handle<YieldTermStructure> flatTermStructure(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, riskFreeRate, dayCounter)));
Handle<YieldTermStructure> flatDividendTS(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, dividendYield, dayCounter)));
Handle<BlackVolTermStructure> flatVolTS(
    boost::shared_ptr<BlackVolTermStructure>(
        new BlackConstantVol(settlementDate, volatility, dayCounter)));

std::vector<Date> dates(4);
dates[0] = settlementDate + 1*Months;
dates[1] = exerciseDate;
dates[2] = exerciseDate + 6*Months;
dates[3] = exerciseDate + 12*Months;
std::vector<Real> 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;
Handle<BlackVolTermStructure> blackSurface(
    boost::shared_ptr<BlackVolTermStructure>(new
        BlackVarianceSurface(settlementDate, dates,
                              strikes, vols, dayCounter)));

```

```

boost::shared_ptr<StrikedTypePayoff> payoff(new
    PlainVanillaPayoff(type, strike));

boost::shared_ptr<BlackScholesProcess> stochasticProcess(new
    BlackScholesProcess(
        underlyingH,
        flatDividendTS,
        flatTermStructure,
        flatVolTS));

// European option
VanillaOption euroOption(stochasticProcess, payoff, exercise,
    boost::shared_ptr<PricingEngine>(new AnalyticEuropeanEngine()));

// method: Black Scholes Engine
method = "equivalent European option ";
value = euroOption.NPV();
std::cout << method << " "
    << DecimalFormatter::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(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DecimalFormatter::toString(value, 6) << "\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Binomial Method (CRR)
method = "Binomial Cox-Ross-Rubinstein ";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DecimalFormatter::toString(value, 6) << "\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Equal Probability Additive Binomial Tree (EQP)
method = "Additive equiprobabilities ";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<AdditiveEQPBinomialTree>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DecimalFormatter::toString(value, 6) << "\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

```

```

// Equal Jumps Additive Binomial Tree (Trigeorgis)
method = "Binomial Trigeorgis ";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DecimalFormatter::toString(value, 6) << "\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Tian Binomial Tree (third moment matching)
method = "Binomial Tian ";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<Tian>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DecimalFormatter::toString(value, 6) << "\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Leisen-Reimer Binomial Tree
method = "Binomial Leisen-Reimer ";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DecimalFormatter::toString(value, 6) << "\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Barone-Adesi and Whaley approximation
method = "Barone-Adesi and Whaley approx. ";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BaroneAdesiWhaleyApproximationEngine));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DecimalFormatter::toString(value, 6) << "\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Bjerksund and Stensland approximation
method = "Bjerksund and Stensland approx. ";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BjerksundStenslandApproximationEngine));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << " "
    << DecimalFormatter::toString(value, 6) << "\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::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;
    }
}
```

9.2 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 = 5;

Integer swapLengths[] = {
    1,    2,    3,    4,    5};
Volatility swaptionVols[] = {
    14.90, 13.40, 12.28, 11.89, 11.48,
    12.90, 12.01, 11.46, 11.08, 10.40,
    11.49, 11.12, 10.70, 10.10, 9.57,
    10.47, 10.21, 9.80, 9.51, 12.70,
    10.00, 9.50, 9.00, 12.30, 11.60};

void calibrateModel(const boost::shared_ptr<ShortRateModel>& model,
                   const std::vector<boost::shared_ptr<CalibrationHelper> >&
                                   helpers,
                   Real lambda) {

    Simplex om(lambda, 1e-9);
    om.setEndCriteria(EndCriteria(10000, 1e-7));
    model->calibrate(helpers, om);

    #if defined(QL_PATCH_DARWIN)
    // to be investigated
    return;
    #endif

    // Output the implied Black volatilities
    for (Size i=0; i<numRows; i++) {
        Size j = numCols - i - 1; // 1x5, 2x4, 3x3, 4x2, 5x1
        Size k = i*numCols + j;
        Real npv = helpers[i]->modelValue();
        Volatility implied = helpers[i]->impliedVolatility(npv, 1e-4,
            1000, 0.05, 0.50)*100.0;
        Volatility diff = implied - swaptionVols[k];

        std::cout << i+1 << "x" << swapLengths[j]
            << ": model " << DecimalFormatter::toString(implied,2,5)
            << ", market " << DecimalFormatter::toString(swaptionVols[k],2,5)
            << " (" << DecimalFormatter::toString(diff,2,5) << ")\n";
    }
}

int main(int, char* [])
{
    try {
        QL_IO_INIT

        Date todaysDate(15, February, 2002);
        Calendar calendar = TARGET();
        Date settlementDate(19, February, 2002);
        Settings::instance().setEvaluationDate(todaysDate);

        // flat yield term structure impling 1x5 swap at 5%
        boost::shared_ptr<Quote> flatRate(new SimpleQuote(0.04875825));
        boost::shared_ptr<FlatForward> myTermStructure(
            new FlatForward(settlementDate, Handle<Quote>(flatRate),
```



```

        Actual365Fixed()));
Handle<YieldTermStructure> rhTermStructure;
rhTermStructure.linkTo(myTermStructure);

// Define the ATM/OTM/ITM swaps
Frequency fixedLegFrequency = Annual;
BusinessDayConvention fixedLegConvention = Unadjusted;
BusinessDayConvention floatingLegConvention = ModifiedFollowing;
DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
Frequency floatingLegFrequency = Semiannual;
bool payFixedRate = true;
Integer fixingDays = 2;
Rate dummyFixedRate = 0.03;
boost::shared_ptr<Xibor> indexSixMonths(new
    Euribor(6, Months, rhTermStructure));

Date startDate = calendar.advance(settlementDate, 1, Years,
    floatingLegConvention);
Date maturity = calendar.advance(startDate, 5, Years,
    floatingLegConvention);
Schedule fixedSchedule(calendar, startDate, maturity,
    fixedLegFrequency, fixedLegConvention);
Schedule floatSchedule(calendar, startDate, maturity,
    floatingLegFrequency, floatingLegConvention);
boost::shared_ptr<SimpleSwap> swap(new SimpleSwap(
    payFixedRate, 1000.0,
    fixedSchedule, dummyFixedRate, fixedLegDayCounter,
    floatSchedule, indexSixMonths, fixingDays, 0.0,
    rhTermStructure));
Rate fixedATMRate = swap->fairRate();
Rate fixedOTMRate = fixedATMRate * 1.2;
Rate fixedITMRate = fixedATMRate * 0.8;

boost::shared_ptr<SimpleSwap> atmSwap(new SimpleSwap(
    payFixedRate, 1000.0,
    fixedSchedule, fixedATMRate, fixedLegDayCounter,
    floatSchedule, indexSixMonths, fixingDays, 0.0,
    rhTermStructure));
boost::shared_ptr<SimpleSwap> otmSwap(new SimpleSwap(
    payFixedRate, 1000.0,
    fixedSchedule, fixedOTMRate, fixedLegDayCounter,
    floatSchedule, indexSixMonths, fixingDays, 0.0,
    rhTermStructure));
boost::shared_ptr<SimpleSwap> itmSwap(new SimpleSwap(
    payFixedRate, 1000.0,
    fixedSchedule, fixedITMRate, fixedLegDayCounter,
    floatSchedule, indexSixMonths, fixingDays, 0.0,
    rhTermStructure));

// defining the swaptions to be used in model calibration
std::vector<Period> swaptionMaturities;
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));

std::vector<boost::shared_ptr<CalibrationHelper> > swaptions;

// List of times that have to be included in the timegrid
std::list<Time> times;

Size i;
for (i=0; i<numRows; i++) {
    Size j = numCols - i - 1; // 1x5, 2x4, 3x3, 4x2, 5x1
    Size k = i*numCols + j;
    boost::shared_ptr<Quote> vol(new

```

```

        SimpleQuote(swaptionVols[k]*0.01));
    swaptions.push_back(boost::shared_ptr<CalibrationHelper>(new
        SwaptionHelper(swaptionMaturities[i],
            Period(swapLengths[j], Years),
            Handle<Quote>(vol),
            indexSixMonths,
            rhTermStructure)));
    swaptions.back()->addTimesTo(times);
}

// Building time-grid
TimeGrid grid(times.begin(), times.end(), 30);

// defining the models
boost::shared_ptr<G2> modelG2(new G2(rhTermStructure));
boost::shared_ptr<HullWhite> modelHW(new HullWhite(rhTermStructure));

#define ALSO_NUMERICAL_MODELS

#ifdef ALSO_NUMERICAL_MODELS
    boost::shared_ptr<HullWhite> modelHW2(new HullWhite(rhTermStructure));
    boost::shared_ptr<BlackKarasinski> modelBK(new
        BlackKarasinski(rhTermStructure));
#endif

// model calibrations

std::cout << "G2 (analytic formulae) calibration" << std::endl;
for (i=0; i<swaptions.size(); i++)
    swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
        new G2SwaptionEngine(modelG2, 6.0, 16)));

calibrateModel(modelG2, swaptions, 0.05);
std::cout << "calibrated to:\n"
    << "a = " << modelG2->params()[0] << ", "
    << "sigma = " << modelG2->params()[1] << "\n"
    << "b = " << modelG2->params()[2] << ", "
    << "eta = " << modelG2->params()[3] << "\n"
    << "rho = " << modelG2->params()[4]
    << std::endl << std::endl;

std::cout << "Hull-White (analytic formulae) calibration" << std::endl;
for (i=0; i<swaptions.size(); i++)
    swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
        new JamshidianSwaptionEngine(modelHW)));

calibrateModel(modelHW, swaptions, 0.05);
std::cout << "calibrated to:\n"
    << "a = " << modelHW->params()[0] << ", "
    << "sigma = " << modelHW->params()[1]
    << std::endl << std::endl;

#ifdef ALSO_NUMERICAL_MODELS
    std::cout << "Hull-White (numerical) calibration" << std::endl;
    for (i=0; i<swaptions.size(); i++)
        swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
            new TreeSwaptionEngine(modelHW2, grid)));

    calibrateModel(modelHW2, swaptions, 0.05);
    std::cout << "calibrated to:\n"
        << "a = " << modelHW2->params()[0] << ", "
        << "sigma = " << modelHW2->params()[1]
        << std::endl << std::endl;

```

```

std::cout << "Black-Karasinski (numerical) calibration" << std::endl;
for (i=0; i<swaptions.size(); i++)
    swaptions[i]->setPricingEngine(boost::shared_ptr<PricingEngine>(
        new TreeSwaptionEngine(modelBK,grid)));

calibrateModel(modelBK, swaptions, 0.05);
std::cout << "calibrated to:\n"
    << "a = " << modelBK->params()[0] << ", "
    << "sigma = " << modelBK->params()[1]
    << std::endl << std::endl;
#endif

// ATM Bermudan swaption pricing

std::cout << "Payer bermudan swaption "
    << "struck at " << RateFormatter::toString(fixedATMRate)
    << " (ATM)" << std::endl;

std::vector<Date> bermudanDates;
const std::vector<boost::shared_ptr<CashFlow> >& leg =
    swap->fixedLeg();
for (i=0; i<leg.size(); i++) {
    boost::shared_ptr<Coupon> coupon =
        boost::dynamic_pointer_cast<Coupon>(leg[i]);
    bermudanDates.push_back(coupon->accrualStartDate());
}

boost::shared_ptr<Exercise> bermudaExercise(new
    BermudanExercise(bermudanDates));

Swaption bermudanSwaption(atmSwap, bermudaExercise, rhTermStructure,
    boost::shared_ptr<PricingEngine>());

// Do the pricing for each model

// G2 price the European swaption here, it should switch to bermudan
bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
    TreeSwaptionEngine(modelG2, 50)));
std::cout << "G2:      " << bermudanSwaption.NPV() << std::endl;

bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelHW, 50)));
std::cout << "HW:      " << bermudanSwaption.NPV() << std::endl;

#ifdef ALSO_NUMERICAL_MODELS
    bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
        TreeSwaptionEngine(modelHW2, 50)));
    std::cout << "HW (num): " << bermudanSwaption.NPV() << std::endl;

    bermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(new
        TreeSwaptionEngine(modelBK, 50)));
    std::cout << "BK:      " << bermudanSwaption.NPV() << std::endl;
#endif

// OTM Bermudan swaption pricing

std::cout << "Payer bermudan swaption "
    << "struck at " << RateFormatter::toString(fixedOTMRate)
    << " (OTM)" << std::endl;

Swaption otmBermudanSwaption(otmSwap, bermudaExercise, rhTermStructure,
    boost::shared_ptr<PricingEngine>());

```

```

// Do the pricing for each model
otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelG2, 50)));
std::cout << "G2:      " << otmBermudanSwaption.NPV() << std::endl;

otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelHW, 50)));
std::cout << "HW:      " << otmBermudanSwaption.NPV() << std::endl;

#ifdef ALSO_NUMERICAL_MODELS
otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelHW2, 50)));
std::cout << "HW (num): " << otmBermudanSwaption.NPV() << std::endl;

otmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelBK, 50)));
std::cout << "BK:      " << otmBermudanSwaption.NPV() << std::endl;
#endif

// ITM Bermudan swaption pricing

std::cout << "Payer bermudan swaption "
    << "struck at " << RateFormatter::toString(fixedITMRate)
    << " (ITM)" << std::endl;

Swaption itmBermudanSwaption(itmSwap, bermudaExercise, rhTermStructure,
    boost::shared_ptr<PricingEngine>());

// Do the pricing for each model
itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelG2, 50)));
std::cout << "G2:      " << itmBermudanSwaption.NPV() << std::endl;

itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelHW, 50)));
std::cout << "HW:      " << itmBermudanSwaption.NPV() << std::endl;

#ifdef ALSO_NUMERICAL_MODELS
itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelHW2, 50)));
std::cout << "HW (num): " << itmBermudanSwaption.NPV() << std::endl;

itmBermudanSwaption.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new TreeSwaptionEngine(modelBK, 50)));
std::cout << "BK:      " << itmBermudanSwaption.NPV() << std::endl;
#endif

return 0;
} catch (std::exception& e) {
    std::cout << e.what() << std::endl;
    return 1;
} catch (...) {
    std::cout << "unknown error" << std::endl;
    return 1;
}
}

```

9.3 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.ederman.com/emanuelderman/GSQSpapers/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,
                    Real strike,
                    Real s0,
                    Volatility sigma,
                    Rate r)
    : maturity_(maturity), payoff_(type, strike), s0_(s0),
      sigma_(sigma), r_(r) {

        // value of the option
        DiscountFactor rDiscount = QL_EXP(-r_*maturity_);
        DiscountFactor qDiscount = 1.0;
        Real forward = s0_*qDiscount/rDiscount;
        Real variance = sigma_*sigma_*maturity_;
        boost::shared_ptr<StrikedTypePayoff> payoff(
            new PlainVanillaPayoff(payoff_));
        BlackFormula black(forward,rDiscount,variance,payoff);
        std::cout << "Option value: " << black.value() << std::endl;

        // store option's vega, since Derman and Kamal's formula needs it
        vega_ = black.vega(maturity_);

        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(Size nTimeSteps, Size nSamples);
private:
    Time maturity_;
    PlainVanillaPayoff payoff_;
    Real s0_;
    Volatility sigma_;
    Rate r_;
    Real vega_;
};

// The key for the MonteCarlo simulation is to have a PathPricer that
// implements a value(const Path& path) method.
// This method prices the portfolio for each Path of the random variable
class ReplicationPathPricer : public PathPricer<Path>
{
public:
    // real constructor
    ReplicationPathPricer(Option::Type type,
                          Real underlying,
                          Real strike,
                          Rate r,
                          Time maturity,
                          Volatility sigma)
    : type_(type), underlying_(underlying),
      strike_(strike), r_(r), maturity_(maturity), sigma_(sigma) {
        QL_REQUIRE(strike_ > 0.0, "strike must be positive");
        QL_REQUIRE(underlying_ > 0.0, "underlying must be positive");
        QL_REQUIRE(r_ >= 0.0,
                  "risk free rate (r) must be positive or zero");
        QL_REQUIRE(maturity_ > 0.0, "maturity must be positive");
        QL_REQUIRE(sigma_ >= 0.0,
                  "volatility (sigma) must be positive or zero");
    }

    // The value() method encapsulates the pricing code
    Real operator()(const Path& path) const;

private:
    Option::Type type_;
    Real underlying_, strike_;
    Rate r_;
    Time maturity_;
    Volatility sigma_;
};

// Compute Replication Error as in the Derman and Kamal's research note
int main(int, char* [])
{
    try {
        QL_IO_INIT

        Time maturity = 1.0/12.0;    // 1 month
        Real strike = 100;
    }
}

```

```

    Real underlying = 100;
    Volatility volatility = 0.20; // 20%
    Rate riskFreeRate = 0.05; // 5%
    ReplicationError rp(Option::Call, maturity, strike, underlying,
                       volatility, riskFreeRate);

    Size scenarios = 50000;
    Size 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.
*/
Real 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
    Size n = path.size();
    QL_REQUIRE(n>0, "the path cannot be empty");

    // discrete hedging interval
    Time dt = maturity_/n;

    // For simplicity, we assume the stock pays no dividends.
    Rate stockDividendYield = 0.0;

    // let's start
    Time t = 0;

    // stock value at t=0
    Real stock = underlying_;
    Real stockLogGrowth = 0.0;

    // money account at t=0
    Real money_account = 0.0;

    /******
    *** the initial deal ***
    *****/
    // option fair price (Black-Scholes) at t=0
    DiscountFactor rDiscount = QL_EXP(-r_*maturity_);
    DiscountFactor qDiscount = QL_EXP(-stockDividendYield*maturity_);
    Real forward = stock*qDiscount/rDiscount;
    Real variance = sigma_*sigma_*maturity_;
    boost::shared_ptr<StrikedTypePayoff> payoff(
        new PlainVanillaPayoff(type_, strike_));

```

```

BlackFormula black(forward,rDiscount,variance,payoff);
// sell the option, cash in its premium
money_account += black.value();
// compute delta
Real delta = black.delta(stock);
// delta-hedge the option buying stock
Real stockAmount = delta;
money_account -= stockAmount*stock;

/***** hedging during option life *****/
for (Size 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
    rDiscount = QL_EXP(-r*(maturity_-t));
    qDiscount = QL_EXP(-stockDividendYield*(maturity_-t));
    forward = stock*qDiscount/rDiscount;
    variance = sigma_*sigma_*(maturity_-t);
    BlackFormula black(forward,rDiscount,variance,payoff);

    // recalculate delta
    delta = black.delta(stock);

    // 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
Real 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(Size nTimeSteps, Size nSamples)
{
    QL_REQUIRE(nTimeSteps>0, "the number of steps must be > 0");

```



```

// hedging interval
// Time tau = maturity_ / nTimeSteps;

/* Black-Scholes framework: the underlying stock price evolves
   lognormally with a fixed known volatility that stays constant
   throughout time.
*/
Date today = Date::todaysDate();
DayCounter dayCount = Actual365Fixed();
Handle<Quote> stateVariable(
    boost::shared_ptr<Quote>(new SimpleQuote(s0_)));
Handle<YieldTermStructure> riskFreeRate(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(today, r_, dayCount)));
Handle<YieldTermStructure> dividendYield(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(today, 0.0, dayCount)));
Handle<BlackVolTermStructure> volatility(
    boost::shared_ptr<BlackVolTermStructure>(
        new BlackConstantVol(today, sigma_, dayCount)));
boost::shared_ptr<StochasticProcess> diffusion(
    new BlackScholesProcess(stateVariable, dividendYield,
        riskFreeRate, volatility));

// Black Scholes equation rules the path generator:
// at each step the log of the stock
// will have drift and sigma^2 variance
PseudoRandom::rsg_type rsg =
    PseudoRandom::make_sequence_generator(nTimeSteps, 0);

bool brownianBridge = false;

typedef SingleAsset<PseudoRandom>::path_generator_type generator_type;
boost::shared_ptr<generator_type> myPathGenerator(new
    generator_type(diffusion, maturity_, nTimeSteps,
        rsg, brownianBridge));

// 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
boost::shared_ptr<PathPricer<Path> > myPathPricer(new
    ReplicationPathPricer(payload_.optionType(), s0_,
        payload_.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 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
Real PLMean = MCSimulation.sampleAccumulator().mean();
Real PLStdDev = MCSimulation.sampleAccumulator().standardDeviation();
Real PLSkew = MCSimulation.sampleAccumulator().skewness();
Real PLKurt = MCSimulation.sampleAccumulator().kurtosis();

// Derman and Kamal's formula

```

```
Real theorStd = QL_SQRT(M_PI/4/nTimeSteps)*vega_*sigma_;

std::cout << nSamples << "\t| "
  << nTimeSteps << "\t | "
  << DecimalFormatter::toString(PLMean, 3) << " \t| "
  << DecimalFormatter::toString(PLStdDev, 2) << " \t | "
  << DecimalFormatter::toString(theorStd, 2) << " \t | "
  << DecimalFormatter::toString(PLSkew, 2) << " \t| "
  << DecimalFormatter::toString(PLKurt, 2) << std::endl;
}
```

9.4 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,
                   Real strike,
                   Real s0,
                   Volatility sigma,
                   Rate r,
                   Rate q)
        : type_(type), maturity_(maturity),
          strike_(strike),
          s0_(s0),
          sigma_(sigma), r_(r), q_(q) {}

    Real operator()(Real x) const {
        Real 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_;
    Real strike_;
    Real s0_;
    Volatility sigma_;
    Rate r_,q_;
};

int main(int, char* [])
{
    try {
        QL_IO_INIT

        std::cout << "Using " << QL_VERSION << std::endl << std::endl;

        // our option
        Option::Type type(Option::Call);
        Real underlying = 7;
        Real strike = 8;
        Spread dividendYield = 0.05;
        Rate riskFreeRate = 0.05;

        Date todaysDate(15, May, 1998);
        Date settlementDate(17, May, 1998);
        Settings::instance().setEvaluationDate(todaysDate);

        Date exerciseDate(17, May, 1999);
        DayCounter dayCounter = Actual365Fixed();
        Time maturity = dayCounter.yearFraction(settlementDate,
                                                exerciseDate);

        Volatility 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;

Date midlifeDate(19, November, 1998);
std::vector<Date> exDates(2);
exDates[0]=midlifeDate;
exDates[1]=exerciseDate;

boost::shared_ptr<Exercise> exercise(
    new EuropeanExercise(exerciseDate));
boost::shared_ptr<Exercise> amExercise(
    new AmericanExercise(settlementDate,
        exerciseDate));
boost::shared_ptr<Exercise> berExercise(new BermudanExercise(exDates));

Handle<Quote> underlyingH(
    boost::shared_ptr<Quote>(new SimpleQuote(underlying)));

// bootstrap the yield/dividend/vol curves
Handle<YieldTermStructure> flatTermStructure(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, riskFreeRate, dayCounter)));
Handle<YieldTermStructure> flatDividendTS(
    boost::shared_ptr<YieldTermStructure>(
        new FlatForward(settlementDate, dividendYield, dayCounter)));
Handle<BlackVolTermStructure> flatVolTS(
    boost::shared_ptr<BlackVolTermStructure>(
        new BlackConstantVol(settlementDate, volatility, dayCounter)));

std::vector<Date> dates(4);
dates[0] = settlementDate + 1*Months;
dates[1] = exerciseDate;
dates[2] = exerciseDate + 6*Months;
dates[3] = exerciseDate + 12*Months;
std::vector<Real> 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;

    Handle<BlackVolTermStructure> blackSurface(
        boost::shared_ptr<BlackVolTermStructure>(
            new BlackVarianceSurface(settlementDate, dates,
                                    strikes, vols, dayCounter)));

    boost::shared_ptr<StrikedTypePayoff> payoff(new
        PlainVanillaPayoff(type, strike));

    boost::shared_ptr<BlackScholesProcess> stochasticProcess(new
        BlackScholesProcess(underlyingH, flatDividendTS,
                            flatTermStructure,
                            // blackSurface
                            flatVolTS));

    EuropeanOption option(stochasticProcess, payoff, exercise);

    std::string method;
    Real value, discrepancy, rightValue, relativeDiscrepancy;

    std::cout << std::endl << std::endl;

    // write column headings
    std::cout << "Method\t\tValue\tEstimatedError\tDiscrepancy"
                << "\tRel. Discr." << std::endl;

    // method: Black-Scholes Engine
    method = "Black-Scholes";
    option.setPricingEngine(boost::shared_ptr<PricingEngine>(
        new AnalyticEuropeanEngine()));
    rightValue = value = option.NPV();
    discrepancy = QL_FABS(value-rightValue);
    relativeDiscrepancy = discrepancy/rightValue;
    std::cout << method << "\t"
                << DecimalFormatter::toString(value, 4) << "\t"
                << "N/A\t\t"
                << DecimalFormatter::toString(discrepancy, 6) << "\t"
                << DecimalFormatter::toString(relativeDiscrepancy, 6)
                << std::endl;

    // method: Integral
    method = "Integral";
    option.setPricingEngine(boost::shared_ptr<PricingEngine>(
        new IntegralEngine()));
    value = option.NPV();
    discrepancy = QL_FABS(value-rightValue);
    relativeDiscrepancy = discrepancy/rightValue;
    std::cout << method << "\t"
                << DecimalFormatter::toString(value, 4) << "\t"
                << "N/A\t\t"
                << DecimalFormatter::toString(discrepancy, 6) << "\t"
                << DecimalFormatter::toString(relativeDiscrepancy, 6)
                << std::endl;

/*
    // method: Integral
    method = "Binary Cash";
    option.setPricingEngine(boost::shared_ptr<PricingEngine>(
        new IntegralCashOrNothingEngine(1.0)));
    value = option.NPV();
    discrepancy = QL_FABS(value-rightValue);
    relativeDiscrepancy = discrepancy/rightValue;

```

```

std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// method: Integral
method = "Binary Asset";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new IntegralAssetOrNothingEngine()));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

*/

Size timeSteps = 801;

// Binomial Method (JR)
method = "Binomial (JR)";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<JarrowRudd>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Binomial Method (CRR)
method = "Binomial (CRR)";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<CoxRossRubinstein>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Equal Probability Additive Binomial Tree (EQP)
method = "Additive (EQP)";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<AdditiveEQPBinoimialTree>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

```

```

// Equal Jumps Additive Binomial Tree (Trigeorgis)
method = "Bin. Trigeorgis";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<Trigeorgis>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Tian Binomial Tree (third moment matching)
method = "Binomial Tian";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<Tian>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Leisen-Reimer Binomial Tree
method = "Binomial LR";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new BinomialVanillaEngine<LeisenReimer>(timeSteps)));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

// Finite Differences Method: not implemented

/*method = "Finite Diff.";
option.setPricingEngine(boost::shared_ptr<PricingEngine>(
    new FdVanillaEngine()));
value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;*/

// Monte Carlo Method
timeSteps = 1;

method = "MC (crude)";
Size mcSeed = 42;

boost::shared_ptr<PricingEngine> mcengine1;
mcengine1 =
    MakeMCEuropeanEngine<PseudoRandom>().withStepsPerYear(timeSteps)

```

```

                                .withTolerance(0.02)
                                .withSeed(mcSeed);

option.setPricingEngine(mcengine1);

value = option.NPV();
Real errorEstimate = option.errorEstimate();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << DecimalFormatter::toString(errorEstimate, 4) << "\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::toString(relativeDiscrepancy, 6)
    << std::endl;

method = "MC (Sobol)";
timeSteps = 1;
Size nSamples = 32768; // 2^15

boost::shared_ptr<PricingEngine> mcengine2;
mcengine2 =
    MakeMCEuropeanEngine<LowDiscrepancy>().withStepsPerYear(timeSteps)
                                           .withSamples(nSamples);
option.setPricingEngine(mcengine2);

value = option.NPV();
discrepancy = QL_FABS(value-rightValue);
relativeDiscrepancy = discrepancy/rightValue;
std::cout << method << "\t"
    << DecimalFormatter::toString(value, 4) << "\t"
    << "N/A\t\t"
    << DecimalFormatter::toString(discrepancy, 6) << "\t"
    << DecimalFormatter::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;
}
}

```


9.5 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;
```

9.6 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, char* [])
{
    try {
        QL_IO_INIT

        /*****
         *** MARKET DATA ***
         *****/

        Calendar calendar = TARGET();
        // uncommenting the following line generates an error
        // calendar = Tokyo();
        Date settlementDate(22, September, 2004);
        // must be a business day
        settlementDate = calendar.adjust(settlementDate);

        Integer fixingDays = 2;
        Date todaysDate = calendar.advance(settlementDate, -fixingDays, Days);
        // nothing to do with Date::todaysDate
        Settings::instance().setEvaluationDate(todaysDate);

        todaysDate = Settings::instance().evaluationDate();
        std::cout << "Today: "
            << WeekdayFormatter::toString(todaysDate.weekday())
            << ", " + DateFormatter::toString(todaysDate)
            << std::endl;

        std::cout << "Settlement date: "
            << WeekdayFormatter::toString(settlementDate.weekday())
            << ", " + DateFormatter::toString(settlementDate)
            << std::endl;

        // deposits
        Rate d1wQuote=0.0382;
        Rate d1mQuote=0.0372;
        Rate d3mQuote=0.0363;
        Rate d6mQuote=0.0353;
        Rate d9mQuote=0.0348;
        Rate d1yQuote=0.0345;
        // FRAs
        Rate fra3x6Quote=0.037125;
        Rate fra6x9Quote=0.037125;
        Rate fra6x12Quote=0.037125;
        // futures
        Real fut1Quote=96.2875;
        Real fut2Quote=96.7875;
        Real fut3Quote=96.9875;
        Real fut4Quote=96.6875;
        Real fut5Quote=96.4875;
        Real fut6Quote=96.3875;
        Real fut7Quote=96.2875;
    }
}

```

```

Real fut8Quote=96.0875;
// swaps
Rate s2yQuote=0.037125;
Rate s3yQuote=0.0398;
Rate s5yQuote=0.0443;
Rate s10yQuote=0.05165;
Rate 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
boost::shared_ptr<Quote> d1wRate(new SimpleQuote(d1wQuote));
boost::shared_ptr<Quote> d1mRate(new SimpleQuote(d1mQuote));
boost::shared_ptr<Quote> d3mRate(new SimpleQuote(d3mQuote));
boost::shared_ptr<Quote> d6mRate(new SimpleQuote(d6mQuote));
boost::shared_ptr<Quote> d9mRate(new SimpleQuote(d9mQuote));
boost::shared_ptr<Quote> d1yRate(new SimpleQuote(d1yQuote));
// FRAs
boost::shared_ptr<Quote> fra3x6Rate(new SimpleQuote(fra3x6Quote));
boost::shared_ptr<Quote> fra6x9Rate(new SimpleQuote(fra6x9Quote));
boost::shared_ptr<Quote> fra6x12Rate(new SimpleQuote(fra6x12Quote));
// futures
boost::shared_ptr<Quote> fut1Price(new SimpleQuote(fut1Quote));
boost::shared_ptr<Quote> fut2Price(new SimpleQuote(fut2Quote));
boost::shared_ptr<Quote> fut3Price(new SimpleQuote(fut3Quote));
boost::shared_ptr<Quote> fut4Price(new SimpleQuote(fut4Quote));
boost::shared_ptr<Quote> fut5Price(new SimpleQuote(fut5Quote));
boost::shared_ptr<Quote> fut6Price(new SimpleQuote(fut6Quote));
boost::shared_ptr<Quote> fut7Price(new SimpleQuote(fut7Quote));
boost::shared_ptr<Quote> fut8Price(new SimpleQuote(fut8Quote));
// swaps
boost::shared_ptr<Quote> s2yRate(new SimpleQuote(s2yQuote));
boost::shared_ptr<Quote> s3yRate(new SimpleQuote(s3yQuote));
boost::shared_ptr<Quote> s5yRate(new SimpleQuote(s5yQuote));
boost::shared_ptr<Quote> s10yRate(new SimpleQuote(s10yQuote));
boost::shared_ptr<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();

boost::shared_ptr<RateHelper> d1w(new DepositRateHelper(
    Handle<Quote>(d1wRate),
    1, Weeks, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d1m(new DepositRateHelper(
    Handle<Quote>(d1mRate),
    1, Months, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d3m(new DepositRateHelper(
    Handle<Quote>(d3mRate),

```

```

        3, Months, fixingDays,
        calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d6m(new DepositRateHelper(
    Handle<Quote>(d6mRate),
    6, Months, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d9m(new DepositRateHelper(
    Handle<Quote>(d9mRate),
    9, Months, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));
boost::shared_ptr<RateHelper> d1y(new DepositRateHelper(
    Handle<Quote>(d1yRate),
    1, Years, fixingDays,
    calendar, ModifiedFollowing, depositDayCounter));

// setup FRAs
boost::shared_ptr<RateHelper> fra3x6(new FraRateHelper(
    Handle<Quote>(fra3x6Rate),
    3, 6, fixingDays, calendar, ModifiedFollowing,
    depositDayCounter));
boost::shared_ptr<RateHelper> fra6x9(new FraRateHelper(
    Handle<Quote>(fra6x9Rate),
    6, 9, fixingDays, calendar, ModifiedFollowing,
    depositDayCounter));
boost::shared_ptr<RateHelper> fra6x12(new FraRateHelper(
    Handle<Quote>(fra6x12Rate),
    6, 12, fixingDays, calendar, ModifiedFollowing,
    depositDayCounter));

// setup futures
Integer futMonths = 3;
Date imm = Date::nextIMMdate(settlementDate);
boost::shared_ptr<RateHelper> fut1(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut2(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut3(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut4(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut5(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,
    futMonths, calendar, ModifiedFollowing,
    depositDayCounter));
imm = Date::nextIMMdate(imm+1);
boost::shared_ptr<RateHelper> fut6(new FuturesRateHelper(
    Handle<Quote>(fut1Price),
    imm,

```

```

        futMonths, calendar, ModifiedFollowing,
        depositDayCounter));
    imm = Date::nextIMMdate(imm+1);
    boost::shared_ptr<RateHelper> fut7(new FuturesRateHelper(
        Handle<Quote>(fut1Price),
        imm,
        futMonths, calendar, ModifiedFollowing,
        depositDayCounter));
    imm = Date::nextIMMdate(imm+1);
    boost::shared_ptr<RateHelper> fut8(new FuturesRateHelper(
        Handle<Quote>(fut1Price),
        imm,
        futMonths, calendar, ModifiedFollowing,
        depositDayCounter));

// setup swaps
Frequency swFixedLegFrequency = Annual;
BusinessDayConvention swFixedLegConvention = Unadjusted;
DayCounter swFixedLegDayCounter = Thirty360(Thirty360::European);
Frequency swFloatingLegFrequency = Semiannual;

boost::shared_ptr<RateHelper> s2y(new SwapRateHelper(
    Handle<Quote>(s2yRate),
    2, Years, fixingDays,
    calendar, swFixedLegFrequency,
    swFixedLegConvention, swFixedLegDayCounter,
    swFloatingLegFrequency, ModifiedFollowing));
boost::shared_ptr<RateHelper> s3y(new SwapRateHelper(
    Handle<Quote>(s3yRate),
    3, Years, fixingDays,
    calendar, swFixedLegFrequency,
    swFixedLegConvention, swFixedLegDayCounter,
    swFloatingLegFrequency, ModifiedFollowing));
boost::shared_ptr<RateHelper> s5y(new SwapRateHelper(
    Handle<Quote>(s5yRate),
    5, Years, fixingDays,
    calendar, swFixedLegFrequency,
    swFixedLegConvention, swFixedLegDayCounter,
    swFloatingLegFrequency, ModifiedFollowing));
boost::shared_ptr<RateHelper> s10y(new SwapRateHelper(
    Handle<Quote>(s10yRate),
    10, Years, fixingDays,
    calendar, swFixedLegFrequency,
    swFixedLegConvention, swFixedLegDayCounter,
    swFloatingLegFrequency, ModifiedFollowing));
boost::shared_ptr<RateHelper> s15y(new SwapRateHelper(
    Handle<Quote>(s15yRate),
    15, Years, fixingDays,
    calendar, swFixedLegFrequency,
    swFixedLegConvention, swFixedLegDayCounter,
    swFloatingLegFrequency, ModifiedFollowing));

/*****
** CURVE BUILDING **
*****/

// Any DayCounter would be fine.
// ActualActual::ISDA ensures that 30 years is 30.0
DayCounter termStructureDayCounter =
    ActualActual(ActualActual::ISDA);

double tolerance = 1.0e-15;

// A depo-swap curve

```

```

std::vector<boost::shared_ptr<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);
boost::shared_ptr<YieldTermStructure> depoSwapTermStructure(new
    PiecewiseFlatForward(settlementDate, depoSwapInstruments,
        termStructureDayCounter, tolerance));

// A depo-futures-swap curve
std::vector<boost::shared_ptr<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);
boost::shared_ptr<YieldTermStructure> depoFutSwapTermStructure(new
    PiecewiseFlatForward(settlementDate, depoFutSwapInstruments,
        termStructureDayCounter, tolerance));

// A depo-FRA-swap curve
std::vector<boost::shared_ptr<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);
boost::shared_ptr<YieldTermStructure> depoFRASwapTermStructure(new
    PiecewiseFlatForward(settlementDate, depoFRASwapInstruments,
        termStructureDayCounter, tolerance));

// Term structures that will be used for pricing:
// the one used for discounting cash flows
Handle<YieldTermStructure> discountingTermStructure;
// the one used for forward rate forecasting
Handle<YieldTermStructure> forecastingTermStructure;

/*****
* SWAPS TO BE PRICED *
*****/

```

```

// constant nominal 1,000,000 Euro
Real nominal = 1000000.0;
// fixed leg
Frequency fixedLegFrequency = Annual;
BusinessDayConvention fixedLegConvention = Unadjusted;
BusinessDayConvention floatingLegConvention = ModifiedFollowing;
DayCounter fixedLegDayCounter = Thirty360(Thirty360::European);
Rate fixedRate = 0.04;

// floating leg
Frequency floatingLegFrequency = Semiannual;
boost::shared_ptr<Xibor> euriborIndex(new Euribor(6, Months,
    forecastingTermStructure)); // using the forecasting curve
Spread spread = 0.0;

Integer lenghtInYears = 5;
bool payFixedRate = true;

Date maturity = calendar.advance(settlementDate, lenghtInYears, Years,
    floatingLegConvention);
Schedule fixedSchedule(calendar, settlementDate, maturity,
    fixedLegFrequency, fixedLegConvention);
Schedule floatSchedule(calendar, settlementDate, maturity,
    floatingLegFrequency, floatingLegConvention);
SimpleSwap spot5YearSwap(
    payFixedRate, nominal,
    fixedSchedule, fixedRate, fixedLegDayCounter,
    floatSchedule, euriborIndex, fixingDays, spread,
    discountingTermStructure);

Date fwdStart = calendar.advance(settlementDate, 1, Years);
Date fwdMaturity = calendar.advance(fwdStart, lenghtInYears, Years,
    floatingLegConvention);
Schedule fwdFixedSchedule(calendar, fwdStart, fwdMaturity,
    fixedLegFrequency, fixedLegConvention);
Schedule fwdFloatSchedule(calendar, fwdStart, fwdMaturity,
    floatingLegFrequency, floatingLegConvention);
SimpleSwap oneYearForward5YearSwap(
    payFixedRate, nominal,
    fwdFixedSchedule, fixedRate, fixedLegDayCounter,
    fwdFloatSchedule, euriborIndex, fixingDays, spread,
    discountingTermStructure);

/*****
 * 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()) << std::endl;

```

```

std::cout << dblrule << std::endl;

std::cout << tab << "5-years swap paying "
    << RateFormatter::toString(fixedRate) << std::endl;
std::cout << headers[0] << separator
    << headers[1] << separator
    << headers[2] << separator
    << headers[3] << separator << std::endl;
std::cout << rule << std::endl;

Real 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())
    << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate) << 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 by "
    + RateFormatter::toString(QL_FABS(fairRate-s5yQuote)));

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())
    << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate) << 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())
<< DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
<< RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
<< RateFormatter::toString(fairRate) << 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) << 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())
<< DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
<< RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
<< RateFormatter::toString(fairRate) << 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())
<< DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
<< RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
<< RateFormatter::toString(fairRate) << 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())
    << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate) << 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.

boost::shared_ptr<SimpleQuote> fiveYearsRate =
    boost::dynamic_pointer_cast<SimpleQuote>(s5yRate);
fiveYearsRate->setValue(0.0460);

std::cout << dblrule << std::endl;
std::cout << "5-year market swap-rate = "
    << RateFormatter::toString(s5yRate->value()) << std::endl;
std::cout << dblrule << std::endl;

std::cout << tab << "5-years swap paying "
    << RateFormatter::toString(fixedRate) << 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())
    << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
    << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
    << RateFormatter::toString(fairRate) << 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())
  << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
  << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
  << RateFormatter::toString(fairRate) << 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())
  << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
  << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
  << RateFormatter::toString(fairRate) << 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())
  << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
  << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
  << RateFormatter::toString(fairRate) << 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())
  << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
  << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
  << RateFormatter::toString(fairRate) << 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())
  << DecimalFormatter::toString(NPV,2) << separator;
std::cout << std::setw(headers[2].size())
  << RateFormatter::toString(fairSpread) << separator;
std::cout << std::setw(headers[3].size())
  << RateFormatter::toString(fairRate) << 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 10

Test List

Class **ActualActual**

the correctness of the results is checked against known good values.

Class **AnalyticBarrierEngine**

the correctness of the returned value is tested by reproducing results available in literature.

Class **AnalyticCliquetEngine**

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **AnalyticContinuousGeometricAveragePriceAsianEngine**

- a) the correctness of the returned value is tested by reproducing results available in literature, and results obtained using a discrete average approximation.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class **AnalyticDigitalAmericanEngine**

- a) the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
- b) the correctness of the returned value in case of asset-or-nothing at-hit digital payoff is tested by reproducing results available in literature.
- c) the correctness of the returned value in case of cash-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
- d) the correctness of the returned value in case of asset-or-nothing at-expiry digital payoff is tested by reproducing results available in literature.
- e) the correctness of the returned greeks in case of cash-or-nothing at-hit digital payoff is tested by reproducing numerical derivatives.

Class **AnalyticDiscreteGeometricAveragePriceAsianEngine**

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the available greeks is tested against numerical calculations.

Class **AnalyticDividendEuropeanEngine**

the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class [AnalyticEuropeanEngine](#)

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing results available in literature.
- c) the correctness of the returned greeks is tested by reproducing numerical derivatives.
- d) the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- e) the implied-volatility calculation is tested by checking that it does not modify the option.
- f) the correctness of the returned value in case of cash-or-nothing digital payoff is tested by reproducing results available in literature.
- g) the correctness of the returned value in case of asset-or-nothing digital payoff is tested by reproducing results available in literature.
- h) the correctness of the returned value in case of gap digital payoff is tested by reproducing results available in literature.
- i) the correctness of the returned greeks in case of cash-or-nothing digital payoff is tested by reproducing numerical derivatives.

Class [AnalyticPerformanceEngine](#)

the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class [BaroneAdesiWhaleyApproximationEngine](#)

the correctness of the returned value is tested by reproducing results available in literature.

Class [BinomialVanillaEngine](#)

the correctness of the returned value is tested by checking it against analytic results.

Class [Bichapter](#)

the correctness of the returned values is tested by checking them against known good results.

Class [BivariateCumulativeNormalDistribution](#)

the correctness of the returned value is tested by checking it against known good results.

Class [Bjerk SundStenslandApproximationEngine](#)

the correctness of the returned value is tested by reproducing results available in literature.

Class [Bond](#)

- a) price/yield calculations are cross-checked for consistency.
- b) price/yield calculations are checked against known good values.

Class [Brent](#)

the correctness of the returned values is tested by checking them against known good results.

Class [Calendar](#)

the methods for adding and removing holidays are tested by inspecting the calendar before and after their invocation.

Class CapFloor

- a) the correctness of the returned value is tested by checking that the price of a cap (resp. floor) decreases (resp. increases) with the strike rate.
- b) the relationship between the values of caps, floors and the resulting collars is checked.
- c) the put-call parity between the values of caps, floors and swaps is checked.
- d) the correctness of the returned implied volatility is tested by using it for reproducing the target value.
- e) the correctness of the returned value is tested by checking it against a known good value.

Class CompositeQuote

the correctness of the returned values is tested by checking them against numerical calculations.

Class CompoundForward

- a) the correctness of the curve is tested by reproducing the input data.
- b) the correctness of the curve is tested by checking the consistency between returned rates and swaps priced on the curve.

Class CovarianceDecomposition

cross checked with getCovariance

Class CubicSpline

the correctness of the returned values is tested by reproducing results available in literature.

Class CumulativePoissonDistribution

the correctness of the returned value is tested by checking it against known good results.

Class Date

self-consistency of dates, serial numbers, days of month, months, and weekdays is checked over the whole date range.

Class DerivedQuote

the correctness of the returned values is tested by checking them against numerical calculations.

Class DPlusDMinus

the correctness of the returned values is tested by checking them against numerical calculations.

Class DZero

the correctness of the returned values is tested by checking them against numerical calculations.

Class ExchangeRate

application of direct and derived exchange rate is tested against calculations.

Class [ExchangeRateManager](#)

lookup of direct, triangulated, and derived exchange rates is tested.

Class [Factorial](#)

the correctness of the returned value is tested by checking it against numerical calculations.

Class [FalsePosition](#)

the correctness of the returned values is tested by checking them against known good results.

Class [FaureRsg](#)

the correctness of the returned values is tested by reproducing known good values.

Class [ForwardEngine](#)

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class [ForwardPerformanceEngine](#)

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class [ForwardSpreadedTermStructure](#)

- a) the correctness of the returned values is tested by checking them against numerical calculations.
- b) observability against changes in the underlying term structure and in the added spread is checked.

Class [GammaFunction](#)

the correctness of the returned value is tested by checking it against known good results.

Class [Germany](#)

the correctness of the returned results is tested against a list of known holidays.

Class [HaltonRsg](#)

- a) the correctness of the returned values is tested by reproducing known good values.
- b) the correctness of the returned values is tested by checking their discrepancy against known good values.

Class [ImpliedTermStructure](#)

- a) the correctness of the returned values is tested by checking them against numerical calculations.
- b) observability against changes in the underlying term structure is checked.

Class [InArrearIndexedCoupon](#)

The class is tested by comparing the value of an in-arrear swap against a known good value.

Class [Instrument](#)

observability of class instances is checked.

Class [InterestRate](#)

Converted rates are checked against known good results

Class [InverseCumulativePoisson](#)

the correctness of the returned value is tested by checking it against known good results.

Class [Italy](#)

the correctness of the returned results is tested against a list of known holidays.

Class [JointCalendar](#)

the correctness of the returned results is tested by reproducing the calculations.

Class [JumpDiffusionEngine](#)

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class [JuQuadraticApproximationEngine](#)

the correctness of the returned value is tested by reproducing results available in literature.

Class [KronrodIntegral](#)

the correctness of the result is tested by checking it against known good values.

Member [pseudoSqrt\(const Matrix &, SalvagingAlgorithm::Type\)](#)

- a) the correctness of the results is tested by reproducing known good data.
- b) the correctness of the results is tested by checking returned values against numerical calculations.

Class [MCBarrierEngine](#)

the correctness of the returned value is tested by reproducing results available in literature.

Class [MCBasketEngine](#)

the correctness of the returned value is tested by reproducing results available in literature.

Class [MCDigitalEngine](#)

the correctness of the returned value in case of cash-or-nothing at-hit digital payoff is tested by reproducing known good results.

Class [MCDiscreteArithmeticAPEngine](#)

the correctness of the returned value is tested by reproducing results available in literature.

Class [MCDiscreteGeometricAPEngine](#)

the correctness of the returned value is tested by reproducing results available in literature.

Class [MCEuropeanEngine](#)

the correctness of the returned value is tested by checking it against analytic results.

Class [MersenneTwisterUniformRng](#)

the correctness of the returned values is tested by checking them against known good results.

Class [Money](#)

money arithmetic is tested with and without currency conversions.

Class [MultiCubicSpline](#)

interpolated values are checked against the original function.

Class [Newton](#)

the correctness of the returned values is tested by checking them against known good results.

Class [NewtonSafe](#)

the correctness of the returned values is tested by checking them against known good results.

Class [NormalDistribution](#)

the correctness of the returned value is tested by checking it against numerical calculations.
Cross-checks are also performed against the CumulativeNormalDistribution and Inverse-CumulativeNormal classes.

Class [PiecewiseFlatForward](#)

- a) the correctness of the returned values is tested by checking them against the original inputs.
- b) the observability of the term structure is tested.

Class [PoissonDistribution](#)

the correctness of the returned value is tested by checking it against known good results.

Class [QuantoEngine](#)

- a) the correctness of the returned value is tested by reproducing results available in literature.
- b) the correctness of the returned greeks is tested by reproducing numerical derivatives.

Class [Quote](#)

the observability of class instances is tested.

Class [RamdomizedLDS](#)

correct initialization is tested.

Class [Ridder](#)

the correctness of the returned values is tested by checking them against known good results.

Class [Rounding](#)

the correctness of the returned values is tested by checking them against known good results.

Class [Secant](#)

the correctness of the returned values is tested by checking them against known good results.

Class [SeedGenerator](#)

correct initializaion of the single instance is tested.

Class [SegmentIntegral](#)

the correctness of the result is tested by checking it against known good values.

Class [SequenceStatistics](#)

the correctness of the returned values is tested by checking them against numerical calculations.

Class [SimpleDayCounter](#)

the correctness of the results is checked against known good values.

Class [SimpleSwap](#)

- a) the correctness of the returned value is tested by checking that the price of a swap paying the fair fixed rate is null.
- b) the correctness of the returned value is tested by checking that the price of a swap receiving the fair floating-rate spread is null.
- c) the correctness of the returned value is tested by checking that the price of a swap decreases with the paid fixed rate.
- d) the correctness of the returned value is tested by checking that the price of a swap increases with the received floating-rate spread.
- e) the correctness of the returned value is tested by checking it against a known good value.

Class [SimpsonIntegral](#)

the correctness of the result is tested by checking it against known good values.

Class [SobolRsg](#)

- a) the correctness of the returned values is tested by reproducing known good values.
- b) the correctness of the returned values is tested by checking their discrepancy against known good values.

Class [StulzEngine](#)

the correctness of the returned value is tested by reproducing results available in literature.

Class SVD

the correctness of the returned values is tested by checking their properties.

Class Swaption

- a) the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption decreases (resp. increases) with the strike.
- b) the correctness of the returned value is tested by checking that the price of a payer (resp. receiver) swaption increases (resp. decreases) with the spread.
- c) the correctness of the returned value is tested by checking it against that of a swaption on a swap with no spread and a correspondingly adjusted fixed rate.
- d) the correctness of the returned value is tested by checking it against a known good value.

Class SymmetricSchurDecomposition

the correctness of the returned values is tested by checking their properties.

Class TARGET

the correctness of the returned results is tested against a list of known holidays.

Class TrapezoidIntegral

the correctness of the result is tested by checking it against known good values.

Class UnitedKingdom

the correctness of the returned results is tested against a list of known holidays.

Class UnitedStates

the correctness of the returned results is tested against a list of known holidays.

Class YieldTermStructure

observability against evaluation date changes is checked.

Class ZeroSpreadedTermStructure

- a) the correctness of the returned values is tested by checking them against numerical calculations.
- b) observability against changes in the underlying term structure and in the added spread is checked.

Chapter 11

Deprecated List

Member [AffineTermStructure](#)(const Date &today'sDate, const Date &referenceDate, const boost::shared_ptr< AffineTermStructure > &discountTermStructure) const
use the constructor without today's date.

Member [AffineTermStructure](#)(const Date &today'sDate, const Date &referenceDate, const boost::shared_ptr< AffineTermStructure > &discountTermStructure) const
use the constructor without today's date.

Member [applyTo](#)(boost::shared_ptr< DiscretizedAsset > asset) const
use adjustValues() on the asset itself

Member [operator<<](#)(std::ostream &, const Array &)
send to the stream the output of ArrayFormatter

Member [BaseTermStructure](#)(const Date &today'sDate, const Date &referenceDate)
use the constructor without today's date; set the evaluation date through Settings::instance().

Member [today'sDate](#)() const
use Settings::instance().evaluationDate().

Member [CapVolatilityVector](#)(const Date &today'sDate, const Calendar &calendar, Integer settlementDays, const Date &referenceDate)
use one of the other constructors

Class [combining_iterator](#)
use a combination of boost::zip_iterator and boost::transform_iterator instead

Member [CompoundForward](#)(const Date &today'sDate, const Date &referenceDate, const std::vector< Date > &dates)
use the constructor without today's date

Class [coupling_iterator](#)
use a combination of boost::zip_iterator and boost::transform_iterator instead

Member `isEndOfMonth()` const

use the static `isEOM()` method instead

Member `lastDayOfMonth()` const

use the static `endOfMonth()` method instead

Member `plusDays(Integer n)` const

use `date + n*Days` instead

Member `plusWeeks(Integer n)` const

use `date + n*Weeks` instead

Member `plusMonths(Integer n)` const

use `date + n*Months` instead

Member `plusYears(Integer n)` const

use `date + n*Years` instead

Member `plus(Integer n, TimeUnit units)` const

use `date + n*units` instead

Member `plus(const Period &)` const

use `date + period` instead

Member `operator<<(std::ostream &, const Date &)`

send to the stream the output of `DateFormatter`

Member `DiscountCurve(const Date &todayDate, const std::vector< Date > &dates, const std::vector< Discount`

use the constructor without today's date.

Class `DiscountStructure`

use `YieldTermStructure` instead

Class `DiscreteGeometricAPO`

use the `DiscreteAveragingAsianOption` instrument with `AnalyticDiscreteGeometricAveragePriceAsianEngine` instead

Member `addTimesTo(std::list< Time > &l)` const

use `mandatoryTimes()` instead.

Member `DiscretizedAsset(const boost::shared_ptr< NumericalMethod > &method)`

use the constructor with no arguments

Member [DiscretizedDiscountBond](#)(const boost::shared_ptr< NumericalMethod > &method)
use the constructor with no arguments

Class [EuroFormatter](#)
use MoneyFormatter instead

Member [ExtendedDiscountCurve](#)(const Date &today'sDate, const std::vector< Date > &dates, const std::vector<
use the constructor without today's date

Class [filtering_iterator](#)
use boost::filter_iterator instead

Member [FlatForward](#)(const Date &today'sDate, const Date &referenceDate, Rate forward, const DayCounter &d
use one of the non-deprecated constructors.

Member [FlatForward](#)(const Date &today'sDate, const Date &referenceDate, const Handle< Quote > &forward, c
use one of the non-deprecated constructors.

Member [fixing\(\)](#) const =0
use rate() instead

Member [ForwardRateStructure](#)(const Date &today'sDate, const Date &referenceDate)
use the constructor without today's date; set the evaluation date through Settings::instance().

Member [isNull\(\)](#) const
use empty() instead

Class [ICGaussianRng](#)
use InverseCumulativeRng instead

Class [ICGaussianRsg](#)
use InverseCumulativeRsg instead

Member [ImpliedTermStructure](#)(const Handle< YieldTermStructure > &, const Date &newToday'sDate, const Da
use the constructor without today's date; set the evaluation date through Settings::instance().

Member [isNull\(\)](#) const
use empty() instead

Class [lowest_category_iterator](#)
no longer needed after deprecation of coupling_iterator

Member **operator<<**(std::ostream &, const Matrix &)
 send to the stream the output of MatrixFormatter

Class **McDiscreteArithmeticAPO**
 use the DiscreteAveragingAsianOption instrument with MCDiscreteArithmeticAPEngine instead

Member **PiecewiseFlatForward**(const Date &today'sDate, const Date &referenceDate, const std::vector< boost::shared_ptr< Rate > > &rates)
 use a constructor without today's date

Member **PiecewiseFlatForward**(const Date &today'sDate, const std::vector< Date > &dates, const std::vector< Rate > &rates)
 use the constructor without today's date

Class **processing_iterator**
 use boost::transform_iterator instead

Member **maturity**() const
 renamed to latestDate()

Member **applyTo**(boost::shared_ptr< DiscretizedAsset > asset) const
 use adjustValues() on the asset itself

Member **applyTo**(boost::shared_ptr< DiscretizedAsset >) const =0
 use adjustValues() on the asset itself

Class **stepping_iterator**
 use step_iterator instead

Member **Xibor**(const std::string &familyName, Integer n, TimeUnit units, Integer settlementDays, CurrencyTag currency)
 use the constructor taking a Currency instance

Class **XiborManager**
 use IndexManager instead

Member **YieldTermStructure**(const Date &today'sDate, const Date &referenceDate)
 use the constructor without today's date; set the evaluation date through Settings::instance().

Member **zeroYield**(const Date &, bool extrapolate=false) const
 use zeroRate(const Date& d, const DayCounter& dc, Continuous, NoFrequency, bool extrapolate) instead

Member **zeroYield**(Time t, bool extrapolate=false) const

use zeroRate(Time t, Continuous, NoFrequency, bool extrapolate) instead

Member **zeroCoupon**(const Date &, Integer, bool extrapolate=false) const

use zeroRate(const Date&, const DayCounter&, Compounding, Frequency, bool) instead

Member **zeroCoupon**(Time, Integer, bool extrapolate=false) const

use zeroRate(Time, Compounding, Frequency, bool) instead

Member **compoundForward**(const Date &d1, Integer f, bool extrapolate=false) const

use forwardRate(const Date& d1, const Date& d1, const DayCounter& dc, SimpleThenCompounded, Frequency f, bool extrapolate) instead

Member **compoundForward**(Time t1, Integer f, bool extrapolate=false) const

use forwardRate(Time t1, Time t1, SimpleThenCompounded, Frequency f, bool extrapolate) instead

Member **instantaneousForward**(const Date &, bool extrapolate=false) const

use forwardRate(const Date& d1, const Date& d1, const DayCounter& dc, Continuous, NoFrequency, bool extrapolate) instead

Member **instantaneousForward**(Time, bool extrapolate=false) const

use forwardRate(Time t1, Time t1, Continuous, NoFrequency, bool extrapolate) instead

Member **forward**(const Date &d1, const Date &d2, bool extrapolate=false) const

use forwardRate(const Date& d1, const Date& d2, const DayCounter& dc, Continuous, NoFrequency, bool extrapolate) instead

Member **forward**(Time, Time, bool extrapolate=false) const

use forwardRate(Time t1, Time t2, Continuous, NoFrequency, bool extrapolate) instead

Member **ZeroCurve**(const Date &today'sDate, const std::vector< Date > &dates, const std::vector< Rate > &yield

use the constructor without today's date

Member **ZeroYieldStructure**(const Date &today'sDate, const Date &referenceDate)

use the constructor without today's date; set the evaluation date through Settings::instance().

Member **RelinkableHandle**

renamed to Handle

Member **CurrencyTag**

use Currency instead

Member [Actual365](#)

use ActualActual or Actual365Fixed instead

Member [QL_SPECIALIZE_ITERATOR_TRAITS](#)(T)

no longer needed for the Boost iterator library

Member [QL_REVERSE_ITERATOR](#)(iterator, type)

use boost::reverse_iterator instead

Chapter 12

Todo List

Class [AmericanCondition](#)

unify the intrinsicValues/Payoff thing

Class [AmericanExercise](#)

check that everywhere the American condition is applied from earliestDate and not earlier

Class [AmericanPayoffAtExpiry](#)

calculate greeks

Class [AmericanPayoffAtHit](#)

calculate greeks

Class [AnalyticContinuousGeometricAveragePriceAsianEngine](#)

handle seasoned options

Class [AnalyticDigitalAmericanEngine](#)

add more greeks (as of now only delta and rho available)

Class [AnalyticDiscreteGeometricAveragePriceAsianEngine](#)

implement correct theta, rho, dividend-rho, and vega calculation

Class [AUDLibor](#)

check settlement days

Class [BermudanExercise](#)

it would be nice to have a way for making a Bermudan with one exercise date equivalent to an European

Class [BicubicSpline](#)

revise end conditions

Class [BivariateCumulativeNormalDistribution](#)

check accuracy of this algorithm and compare with: 1) Drezner, Z, (1978), Computation of the bivariate normal integral, Mathematics of Computation 32, pp. 277-279. 2) Drezner, Z. and Wesolowsky, G. O. (1990) 'On the Computation of the Bivariate Normal Integral', Journal of Statistical Computation and Simulation 35, pp. 101-107. 3) Drezner, Z (1992) Computation of the Multivariate Normal Integral, ACM Transactions on Mathematics Software 18, pp. 450-460. 4) Drezner, Z (1994) Computation of the Trivariate Normal Integral, Mathematics of Computation 62, pp. 289-294. 5) Genz, A. (1992) 'Numerical Computation of the Multivariate Normal Probabilities', J. Comput. Graph. Stat. 1, pp. 141-150.

Member [drift](#)(Time t, Real x) const

revise extrapolation

Member [diffusion](#)(Time t, Real x) const

revise extrapolation

Class [BlackVarianceCurve](#)

check time extrapolation

Class [BlackVarianceSurface](#)

check time extrapolation

Member [Side](#)

Generalize for n-dimensional conditions

Class [CADLibor](#)

check settlement days

Class [CapVolatilityVector](#)

either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the length vector but an interpolation pointing to the original ones.

Class [CHFLibor](#)

check settlement days and day-count

Class [CliquetOption](#)

add local/global caps/floors
add accrued coupon and last fixing

Class [ContinuousAveragingAsianOption](#)

add running average

Class [DirichletBC](#)

generalize to time-dependent conditions.

Class [DiscreteGeometricASO](#)

add analytical greeks

Class [EarlyExercise](#)

derive a plain American Exercise class (no earliestDate, no payoffAtExpiry)

Class [ExplicitEuler](#)

add Richardson extrapolation

Class [FraRateHelper](#)

convexity adjustment should be implemented.

Class [GenericRiskStatistics](#)

add historical annualized volatility

Class [IntegralEngine](#)

define tolerance for calculate()

Class [JPYLibor](#)

check settlement days

Class [LogLinearInterpolation](#)

implement primitive, derivative, and secondDerivative functions.

Member [pseudoSqrt](#)(const Matrix &, SalvagingAlgorithm::Type)

implement Hypersphere decomposition: 1) Jäckel "Monte Carlo Methods in Finance", Chapter 6 2) Brigo "A Note on Correlation and Rank Reduction" 3) Rapisarda, Brigo, Mercurio "Parameterizing correlations: a geometric interpretation"
implement Higham algorithm: Higham "Computing the nearest correlation matrix"

Class [McDiscreteArithmeticASO](#)

continous-averaging version

Class [MixedScheme](#)

derive variable theta schemes
introduce multi time-level schemes.

Class [MultiCubicSpline](#)

- fix it for Borland compilation
- allow extrapolation as for the other interpolations
- investigate if and how to implement Hyman filters and different boundary conditions

Class [MultiPath](#)

rename as MultiAssetPath

Class [MultiPathGenerator](#)

why store correlation matrix rather than covariance?

Class [NeumannBC](#)

generalize to time-dependent conditions.

Class [Option::arguments](#)

remove `std::vector<Time> stoppingTimes`

how to handle strike-less option (asian average strike, forward, etc.)?

Class [Path](#)

should Path include the $t=0.0$ point? Alternatively all path pricers must be revisited.

Class [RandomizedLDS](#)

implement the other randomization algorithms

Class [ShoutCondition](#)

unify the `intrinsicValues/Payoff` thing

Class [Solver1D](#)

clean up the interface so that it is clear whether the accuracy is specified for x or $f(x)$.

add target value (now the target value is 0.0)

Class [SwapRateHelper](#)

Currency and dayCounter of Xibor should be added to obtain well define SwapRateHelper

Class [SwaptionVolatilityMatrix](#)

either add correct copy behavior or inhibit copy. Right now, a copied instance would end up with its own copy of the exercise date and length vector but an interpolation pointing to the original ones.

Class [TimeGrid](#)

what was the rationale for limiting the grid to positive times? Investigate and see whether we can use it for negative ones as well.

Class [UnitedKingdom](#)

add LIFFE

Class [YieldTermStructure](#)

add derived class `ParSwapTermStructure` similar to `ZeroYieldTermStructure`, `DiscountStructure`, `ForwardRateStructure`

Class [ZARLibor](#)

check settlement days

Chapter 13

Bug List

Class [BlackFormula](#)

When the variance is null, division by zero occur during the calculation of delta, delta forward, gamma, gamma forward, rho, dividend rho, vega, and strike sensitivity.

Class [BPSBasketCalculator](#)

this class must still be checked. It is not guaranteed to yield the right results.

Class [CompoundForward](#)

swap rates are not reproduced exactly when using indexed coupons. Apparently, some assumption about the swap fixings is hard-coded into the bootstrapping algorithm.
the class does not operate correctly when `QL_DISABLE_DEPRECATED` is defined. Investigation is required.

Class [CoxIngersollRoss](#)

this class was not tested enough to guarantee its functionality.

Class [ExtendedCoxIngersollRoss](#)

this class was not tested enough to guarantee its functionality.

Class [FdDividendAmericanOption](#)

sometimes yields negative vega when deeply in-the-money
method `impliedVolatility()` utterly fails

Class [G2](#)

This class was not tested enough to guarantee its functionality.

Class [HullWhite](#)

When the term structure is relinked, the `r0` parameter of the underlying Vasicek model is not updated.

Class [JuQuadraticApproximationEngine](#)

test fails for Borland compiler

Class [LocalVolSurface](#)

this class is untested, probably unreliable.

Class [MCAmericanBasketEngine](#)

this engine does not yet work for put options. More problems might surface.

Class [MultiCubicSpline](#)

- a) cannot interpolate at the grid points on the boundary surface of the N-dimensional region
- b) it does not compile under Borland

Member [sensitivity\(Integer basis=2\) const](#)

this method must still be checked. It is not guaranteed to yield the right results.

Class [TreeCapFloorEngine](#)

caplets which have already fixed are not included in the cap value.

Index

- ~Error
 - QuantLib::Error, [361](#)
- accruedAmount
 - QuantLib::Bond, [215](#)
- Actual365
 - daycounters, [85](#)
- add
 - QuantLib::ExchangeRateManager, [372](#)
 - QuantLib::GeneralStatistics, [432](#)
 - QuantLib::IncrementalStatistics, [472](#)
- addHoliday
 - QuantLib::Calendar, [234](#)
- addSequence
 - QuantLib::IncrementalStatistics, [472](#)
- addTimesTo
 - QuantLib::DiscretizedAsset, [338](#)
- adjust
 - QuantLib::Calendar, [234](#)
- adjustValues
 - QuantLib::DiscretizedAsset, [337](#)
- advance
 - QuantLib::Calendar, [234](#), [235](#)
- AffineTermStructure
 - QuantLib::AffineTermStructure, [140](#)
- amount
 - QuantLib::CashFlow, [252](#)
 - QuantLib::FixedRateCoupon, [398](#)
 - QuantLib::IndexedCoupon, [475](#)
 - QuantLib::ParCoupon, [643](#)
 - QuantLib::Short, [703](#)
 - QuantLib::SimpleCashFlow, [708](#)
- Annual
 - datetime, [82](#)
- applyAfterApplying
 - QuantLib::BoundaryCondition, [217](#)
- applyAfterSolving
 - QuantLib::BoundaryCondition, [217](#)
- applyBeforeApplying
 - QuantLib::BoundaryCondition, [217](#)
- applyBeforeSolving
 - QuantLib::BoundaryCondition, [217](#)
- applyTo
 - QuantLib::AmericanCondition, [141](#)
 - QuantLib::ShoutCondition, [707](#)
 - QuantLib::StepCondition, [732](#)
- ARS
 - currencies, [78](#)
- Asian option engines, [87](#)
- ATS
 - currencies, [78](#)
- AUD
 - currencies, [78](#)
- AutomatedConversion
 - QuantLib::Money, [584](#)
- averageShortfall
 - QuantLib::GenericRiskStatistics, [437](#)
- Barrier option engines, [88](#)
- BaseCurrencyConversion
 - QuantLib::Money, [584](#)
- BaseTermStructure
 - QuantLib::BaseTermStructure, [173](#)
- basispointsensitivity.hpp
 - BasisPointSensitivityBasket, [860](#)
- BasisPointSensitivityBasket
 - basispointsensitivity.hpp, [860](#)
- Basket option engines, [89](#)
- BDT
 - currencies, [78](#)
- BEF
 - currencies, [78](#)
- beta.hpp
 - incompleteBetaFunction, [971](#)
- BGL
 - currencies, [78](#)
- BicubicSpline
 - QuantLib::BicubicSpline, [183](#)
- BilinearInterpolation
 - QuantLib::BilinearInterpolation, [184](#)
- Bimonthly
 - datetime, [82](#)
- binomialdistribution.hpp
 - PeizerPrattMethod2Inversion, [974](#)
- blackVarianceImpl
 - QuantLib::BlackVolatilityTermStructure, [210](#)
- BlackVarianceTermStructure
 - QuantLib::BlackVarianceTermStructure, [208](#)

- BlackVolatilityTermStructure
 - QuantLib::BlackVolatilityTermStructure, 210
- blackVolImpl
 - QuantLib::BlackVarianceTermStructure, 208
- BlackVolTermStructure
 - QuantLib::BlackVolTermStructure, 213
- BoundaryCondition
 - QuantLib::CubicSpline, 298
- BRL
 - currencies, 78
- BusinessDayConvention
 - datetime, 81
- BYB
 - currencies, 78
- CAD
 - currencies, 78
- calculate
 - QuantLib::Instrument, 479
 - QuantLib::LazyObject, 520
- Calendar
 - QuantLib::Calendar, 233
- Calendars, 83
- calibrate
 - QuantLib::ShortRateModel, 706
- Cap/floor engines, 90
- CapFlatVolatilityStructure
 - capvolstructures.hpp, 857
- CapFlatVolatilityVector
 - capflatvolvector.hpp, 1193
- capflatvolvector.hpp
 - CapFlatVolatilityVector, 1193
- CapletForwardVolatilityStructure
 - capvolstructures.hpp, 857
- CapletVolatilityStructure
 - QuantLib::CapletVolatilityStructure, 247
- CapVolatilityStructure
 - QuantLib::CapVolatilityStructure, 249
- CapVolatilityVector
 - QuantLib::CapVolatilityVector, 250
- capvolstructures.hpp
 - CapFlatVolatilityStructure, 857
 - CapletForwardVolatilityStructure, 857
- cashflowvectors.hpp
 - FloatingRateCouponVector, 862
- Ceiling
 - QuantLib::Rounding, 687
- Character functions, 129
- CHF
 - currencies, 78
- cleanPrice
 - QuantLib::Bond, 215
- Cliquet option engines, 91
- close
 - comparison.hpp, 979
- close_enough
 - comparison.hpp, 979
- Closest
 - QuantLib::Rounding, 687
- CLP
 - currencies, 78
- CNY
 - currencies, 78
- comparison.hpp
 - close, 979
 - close_enough, 979
- compoundFactor
 - QuantLib::InterestRate, 484
- CompoundForward
 - QuantLib::CompoundForward, 272
- compoundForward
 - QuantLib::YieldTermStructure, 814
- compoundForwardImpl
 - QuantLib::CompoundForward, 272
 - QuantLib::DiscountStructure, 328
 - QuantLib::ExtendedDiscountCurve, 380
 - QuantLib::ForwardRateStructure, 409
 - QuantLib::ZeroYieldStructure, 824
- ConversionType
 - QuantLib::Money, 584
- COP
 - currencies, 78
- correlationMatrix
 - QuantLib::CovarianceDecomposition, 290
- Coupon
 - QuantLib::Coupon, 289
- CovarianceDecomposition
 - QuantLib::CovarianceDecomposition, 290
- CubicSpline
 - QuantLib::CubicSpline, 298
- currencies
 - ARS, 78
 - ATS, 78
 - AUD, 78
 - BDT, 78
 - BEF, 78
 - BGL, 78
 - BRL, 78
 - BYB, 78
 - CAD, 78
 - CHF, 78
 - CLP, 78

- CNY, [78](#)
- COP, [78](#)
- CurrencyTag, [78](#)
- CYP, [78](#)
- CZK, [78](#)
- DEM, [78](#)
- DKK, [78](#)
- EEK, [78](#)
- EUR, [78](#)
- GBP, [78](#)
- GRD, [78](#)
- HKD, [78](#)
- HUF, [78](#)
- ILS, [78](#)
- INR, [78](#)
- IQD, [78](#)
- IRR, [79](#)
- ISK, [79](#)
- ITL, [79](#)
- JPY, [79](#)
- KRW, [79](#)
- KWD, [79](#)
- LTL, [79](#)
- LVL, [79](#)
- MTL, [79](#)
- MXP, [79](#)
- NOK, [79](#)
- NPR, [79](#)
- NZD, [79](#)
- PKR, [79](#)
- PLN, [79](#)
- ROL, [79](#)
- SAR, [79](#)
- SEK, [79](#)
- SGD, [79](#)
- SIT, [79](#)
- SKK, [79](#)
- THB, [79](#)
- TRL, [79](#)
- TTD, [79](#)
- TWD, [79](#)
- USD, [79](#)
- VEB, [79](#)
- ZAR, [79](#)
- Currencies and FX rates, [77](#)
- Currency
 - QuantLib::Currency, [306](#)
- currency.hpp
 - make_currency, [887](#)
- CurrencyTag
 - currencies, [78](#)
- CYP
 - currencies, [78](#)
- CZK
 - currencies, [78](#)
- Date and time calculations, [80](#)
- datetime
 - Annual, [82](#)
 - Bimonthly, [82](#)
 - BusinessDayConvention, [81](#)
 - EveryFourthMonth, [82](#)
 - Following, [81](#)
 - Frequency, [82](#)
 - ModifiedFollowing, [81](#)
 - ModifiedPreceding, [81](#)
 - MonthEndReference, [81](#)
 - Monthly, [82](#)
 - NoFrequency, [82](#)
 - Once, [82](#)
 - Preceding, [81](#)
 - Quarterly, [82](#)
 - Semiannual, [82](#)
 - Unadjusted, [81](#)
 - Weekday, [82](#)
- Day counters, [85](#)
- DayCounter
 - QuantLib::DayCounter, [317](#)
- daycounters
 - Actual365, [85](#)
- DEFINE_SEQUENCE_STAT_CONST_ -
METHOD_DOUBLE
 - sequencestatistics.hpp, [1009](#)
- DEFINE_SEQUENCE_STAT_CONST_ -
METHOD_VOID
 - sequencestatistics.hpp, [1009](#)
- DEM
 - currencies, [78](#)
- Derived
 - QuantLib::ExchangeRate, [371](#)
- Design patterns, [119](#)
- diffusion
 - QuantLib::BlackScholesProcess, [201](#)
 - QuantLib::Merton76Process, [579](#)
- Direct
 - QuantLib::ExchangeRate, [371](#)
- dirtyPrice
 - QuantLib::Bond, [215](#)
- DiscountCurve
 - QuantLib::DiscountCurve, [326](#)
- discountFactor
 - QuantLib::InterestRate, [484](#)
- discountImpl
 - QuantLib::CompoundForward, [272](#)
 - QuantLib::ForwardRateStructure, [409](#)
 - QuantLib::ZeroYieldStructure, [824](#)
- DiscretizedAsset
 - QuantLib::DiscretizedAsset, [337](#)

- DiscretizedDiscountBond
 - QuantLib::DiscretizedDiscountBond, 339
- DKK
 - currencies, 78
- Down
 - QuantLib::Rounding, 687
- downsideDeviation
 - QuantLib::GenericRiskStatistics, 436
 - QuantLib::IncrementalStatistics, 471
- downsideVariance
 - QuantLib::GenericRiskStatistics, 435
 - QuantLib::IncrementalStatistics, 471
- drift
 - QuantLib::BlackScholesProcess, 201
 - QuantLib::Merton76Process, 579
- EEK
 - currencies, 78
- equivalentRate
 - QuantLib::InterestRate, 485
- Error
 - QuantLib::Error, 361
- errorEstimate
 - QuantLib::GeneralStatistics, 431
 - QuantLib::IncrementalStatistics, 471
- errors.hpp
 - QL_ASSERT, 903
 - QL_ENSURE, 903
 - QL_REQUIRE, 903
- EUR
 - currencies, 78
- Eurex
 - QuantLib::Germany, 441
- evaluationDate
 - QuantLib::Settings, 700
- evaluationDateGuard
 - QuantLib::Settings, 700
- EveryFourthMonth
 - datetime, 82
- evolve
 - QuantLib::BlackScholesProcess, 201
 - QuantLib::Merton76Process, 580
 - QuantLib::StochasticProcess, 736
- Exchange
 - QuantLib::Italy, 501
 - QuantLib::UnitedKingdom, 788
 - QuantLib::UnitedStates, 791
- ExchangeRate
 - QuantLib::ExchangeRate, 371
- expectation
 - QuantLib::EulerDiscretization, 364
 - QuantLib::OrnsteinUhlenbeckProcess, 637
 - QuantLib::StochasticProcess, 736
- expectationValue
 - QuantLib::GeneralStatistics, 432
- expectedShortfall
 - QuantLib::GenericRiskStatistics, 436
- ExtendedDiscountCurve
 - QuantLib::ExtendedDiscountCurve, 380
- Financial instruments, 105
- Finite-differences framework, 96
- FirstDerivative
 - QuantLib::CubicSpline, 298
- firstDerivativeAtCenter
 - valueatcenter.hpp, 923
- fixing
 - QuantLib::FloatingRateCoupon, 402
 - QuantLib::Index, 473
 - QuantLib::IndexedCoupon, 475
 - QuantLib::ParCoupon, 643
 - QuantLib::Xibor, 808
- FlatForward
 - QuantLib::FlatForward, 400
- FloatingRateCouponVector
 - cashflowvectors.hpp, 862
- Floor
 - QuantLib::Rounding, 687
- Following
 - datetime, 81
- format
 - QuantLib::Currency, 306
- formula
 - QuantLib::BlackModel, 197
- forward
 - QuantLib::YieldTermStructure, 814, 815
- Forward option engines, 92
- forwardImpl
 - QuantLib::DiscountStructure, 328
 - QuantLib::ZeroSpreadedTermStructure, 822
 - QuantLib::ZeroYieldStructure, 824
- forwardRate
 - QuantLib::YieldTermStructure, 815
- ForwardRateStructure
 - QuantLib::ForwardRateStructure, 409
- FrankfurtStockExchange
 - QuantLib::Germany, 441
- freeze
 - QuantLib::LazyObject, 520
- Frequency
 - datetime, 82
- gaussianDownsideDeviation
 - QuantLib::GaussianStatistics, 425

- gaussianDownsideVariance
 - QuantLib::GaussianStatistics, 425
- gaussianExpectedShortfall
 - QuantLib::GaussianStatistics, 426
- gaussianPercentile
 - QuantLib::GaussianStatistics, 426
- gaussianPotentialUpside
 - QuantLib::GaussianStatistics, 426
- gaussianRegret
 - QuantLib::GaussianStatistics, 426
- gaussianTopPercentile
 - QuantLib::GaussianStatistics, 426
- gaussianValueAtRisk
 - QuantLib::GaussianStatistics, 426
- GBP
 - currencies, 78
- Generic macros, 124
- getCovariance
 - getcovariance.hpp, 1019
- getcovariance.hpp
 - getCovariance, 1019
- GRD
 - currencies, 78
- Handle
 - QuantLib::Handle, 445
- History
 - QuantLib::History, 449
- HKD
 - currencies, 78
- HUF
 - currencies, 78
- ILS
 - currencies, 78
- impliedRate
 - QuantLib::InterestRate, 484, 485
- ImpliedTermStructure
 - QuantLib::ImpliedTermStructure, 465
- impliedVolatility
 - QuantLib::OneAssetOption, 620
 - QuantLib::SingleAssetOption, 719
- incompleteBetaFunction
 - beta.hpp, 971
- incompletegamma.hpp
 - incompleteGammaFunction, 989
- incompleteGammaFunction
 - incompletegamma.hpp, 989
- indexcashflowvectors.hpp
 - IndexedCouponVector, 867
- IndexedCouponVector
 - indexcashflowvectors.hpp, 867
- INR
 - currencies, 78
- instantaneousForward
 - QuantLib::YieldTermStructure, 814
- IQD
 - currencies, 78
- IRR
 - currencies, 79
- isBusinessDay
 - QuantLib::Calendar, 234
- isEndOfMonth
 - QuantLib::Calendar, 234
 - QuantLib::Date, 312
- isHoliday
 - QuantLib::Calendar, 234
- ISK
 - currencies, 79
- isNull
 - QuantLib::Handle, 445
 - QuantLib::Link, 532
- isOnTime
 - QuantLib::DiscretizedAsset, 338
- Iterator support, 132
- iteratorMacros
 - QL_FULL_ITERATOR_SUPPORT, 132
 - QL_REVERSE_ITERATOR, 132
 - QL_SPECIALIZE_ITERATOR_TRAITS, 132
- ITL
 - currencies, 79
- itmAssetProbability
 - QuantLib::BlackFormula, 194
- itmCashProbability
 - QuantLib::BlackFormula, 194
- itmProbability
 - QuantLib::BlackModel, 198
- JPY
 - currencies, 79
- KnuthUniformRng
 - QuantLib::KnuthUniformRng, 511
- KRW
 - currencies, 79
- kurtosis
 - QuantLib::GeneralStatistics, 431
 - QuantLib::IncrementalStatistics, 472
- KWD
 - currencies, 79
- Lagrange
 - QuantLib::CubicSpline, 298
- lastDayOfMonth
 - QuantLib::Date, 312
- latestDate
 - QuantLib::DepositRateHelper, 321

- QuantLib::FraRateHelper, 415
- QuantLib::FuturesRateHelper, 418
- QuantLib::RateHelper, 681
- QuantLib::SwapRateHelper, 748
- Lattice methods, 108
- LecuyerUniformRng
 - QuantLib::LecuyerUniformRng, 523
- limitMacros
 - QL_EPSILON, 126
 - QL_MAX_INTEGER, 126
 - QL_MAX_REAL, 126
 - QL_MIN_INTEGER, 126
 - QL_MIN_POSITIVE_REAL, 126
 - QL_MIN_REAL, 126
- LinearInterpolation
 - QuantLib::LinearInterpolation, 528
- Link
 - QuantLib::Link, 531
- linkTo
 - QuantLib::Handle, 445
 - QuantLib::Link, 532
- localVolImpl
 - QuantLib::LocalVolCurve, 535
- LocalVolTermStructure
 - QuantLib::LocalVolTermStructure, 539
- LogLinearInterpolation
 - QuantLib::LogLinearInterpolation, 541
- lookup
 - QuantLib::ExchangeRateManager, 372
- LTL
 - currencies, 79
- LVL
 - currencies, 79
- make_currency
 - currency.hpp, 887
- mandatoryTimes
 - QuantLib::DiscretizedAsset, 337
 - QuantLib::DiscretizedOption, 340
- Market
 - QuantLib::Germany, 441
 - QuantLib::Italy, 501
 - QuantLib::UnitedKingdom, 788
 - QuantLib::UnitedStates, 791
- Math functions, 125
- Math tools, 111
- maturity
 - QuantLib::RateHelper, 681
- max
 - QuantLib::GeneralStatistics, 432
 - QuantLib::IncrementalStatistics, 472
- mean
 - QuantLib::GeneralStatistics, 431
 - QuantLib::IncrementalStatistics, 471
- MersenneTwisterUniformRng
 - QuantLib::MersenneTwisterUniformRng, 578
- min
 - QuantLib::GeneralStatistics, 431
 - QuantLib::IncrementalStatistics, 472
- Min and max functions, 130
- miscMacros
 - QL_DUMMY_RETURN, 124
 - QL_IO_INIT, 124
- ModifiedFollowing
 - datetime, 81
- ModifiedPreceding
 - datetime, 81
- MonotonicCubicSpline
 - QuantLib::MonotonicCubicSpline, 586
- Monte Carlo framework, 113
- MonthEndReference
 - datetime, 81
- Monthly
 - datetime, 82
- MTL
 - currencies, 79
- MXP
 - currencies, 79
- name
 - QuantLib::Calendar, 234
 - QuantLib::DayCounter, 317
 - QuantLib::Index, 473
 - QuantLib::Xibor, 808
- NaturalCubicSpline
 - QuantLib::NaturalCubicSpline, 599
- NaturalMonotonicCubicSpline
 - QuantLib::NaturalMonotonicCubicSpline, 600
- next
 - QuantLib::KnuthUniformRng, 511
 - QuantLib::LecuyerUniformRng, 523
 - QuantLib::MersenneTwisterUniformRng, 578
- nextIMMdate
 - QuantLib::Date, 313
- nextRandomizer
 - QuantLib::RamdomizedLDS, 677
- nextWeekday
 - QuantLib::Date, 313
- NoConversion
 - QuantLib::Money, 584
- NoFrequency
 - datetime, 82
- NOK
 - currencies, 79
- None

- QuantLib::Rounding, 687
- NotAKnot
 - QuantLib::CubicSpline, 298
- notifyObservers
 - QuantLib::Observable, 616
- NPR
 - currencies, 79
- nthWeekday
 - QuantLib::Date, 313
- Numeric limits, 126
- Numeric types, 75
- NZD
 - currencies, 79
- Once
 - datetime, 82
- operator+=
 - QuantLib::Matrix, 549
- operator<<
 - QuantLib::Array, 158
 - QuantLib::Date, 314
 - QuantLib::Matrix, 549
- operator==
 - QuantLib::Calendar, 235
 - QuantLib::DayCounter, 317
- parRate
 - QuantLib::YieldTermStructure, 815
- partialRollback
 - QuantLib::Lattice, 516
 - QuantLib::NumericalMethod, 613
- PeizerPrattMethod2Inversion
 - binomialdistribution.hpp, 974
- percentile
 - QuantLib::GeneralStatistics, 432
- performCalculations
 - QuantLib::BarrierOption, 169
 - QuantLib::Bond, 215
 - QuantLib::ForwardVanillaOption, 413
 - QuantLib::Instrument, 479
 - QuantLib::LazyObject, 520
 - QuantLib::MultiAssetOption, 592
 - QuantLib::OneAssetOption, 620
 - QuantLib::OneAssetStrikedOption, 624
 - QuantLib::QuantoVanillaOption, 674
 - QuantLib::Stock, 738
 - QuantLib::Swap, 746
- Periodic
 - QuantLib::CubicSpline, 298
- PiecewiseFlatForward
 - QuantLib::PiecewiseFlatForward, 652, 653
- PKR
 - currencies, 79
- PLN
 - currencies, 79
- plus
 - QuantLib::Date, 313
- plusDays
 - QuantLib::Date, 312
- plusMonths
 - QuantLib::Date, 313
- plusWeeks
 - QuantLib::Date, 313
- plusYears
 - QuantLib::Date, 313
- PoissonPseudoRandom
 - ql/RandomNumbers/rngtraits.hpp, 1136
- postAdjustValues
 - QuantLib::DiscretizedAsset, 337
- postAdjustValuesImpl
 - QuantLib::DiscretizedAsset, 338
 - QuantLib::DiscretizedOption, 341
- potentialUpside
 - QuantLib::GenericRiskStatistics, 436
- preAdjustValues
 - QuantLib::DiscretizedAsset, 337
- preAdjustValuesImpl
 - QuantLib::DiscretizedAsset, 338
- Preceding
 - datetime, 81
- Pricing engines, 86
- PseudoRandom
 - ql/RandomNumbers/rngtraits.hpp, 1136
- pseudoSqrt
 - QuantLib::Matrix, 549
- ql/argsandresults.hpp, 827
- ql/basetermstructure.hpp, 829
- ql/basicdataformatters.hpp, 830
- ql/calendar.hpp, 831
- ql/Calendars/beijing.hpp, 832
- ql/Calendars/budapest.hpp, 833
- ql/Calendars/copenhagen.hpp, 834
- ql/Calendars/germany.hpp, 835
- ql/Calendars/helsinki.hpp, 836
- ql/Calendars/hongkong.hpp, 837
- ql/Calendars/italy.hpp, 838
- ql/Calendars/johannesburg.hpp, 839
- ql/Calendars/jointcalendar.hpp, 840
- ql/Calendars/nullcalendar.hpp, 841
- ql/Calendars/oslo.hpp, 842
- ql/Calendars/riyadh.hpp, 843
- ql/Calendars/seoul.hpp, 844
- ql/Calendars/singapore.hpp, 845
- ql/Calendars/stockholm.hpp, 846

- ql/Calendars/sydney.hpp, 847
- ql/Calendars/taiwan.hpp, 848
- ql/Calendars/target.hpp, 849
- ql/Calendars/tokyo.hpp, 850
- ql/Calendars/toronto.hpp, 851
- ql/Calendars/unitedkingdom.hpp, 852
- ql/Calendars/unitedstates.hpp, 853
- ql/Calendars/warsaw.hpp, 854
- ql/Calendars/wellington.hpp, 855
- ql/Calendars/zurich.hpp, 856
- ql/capvolstructures.hpp, 857
- ql/cashflow.hpp, 859
- ql/CashFlows/basispointsensitivity.hpp, 860
- ql/CashFlows/cashflowvectors.hpp, 861
- ql/CashFlows/coupon.hpp, 863
- ql/CashFlows/fixedratecoupon.hpp, 864
- ql/CashFlows/floatingratecoupon.hpp, 865
- ql/CashFlows/inarrearindexedcoupon.hpp, 866
- ql/CashFlows/indexcashflowvectors.hpp, 867
- ql/CashFlows/indexedcoupon.hpp, 869
- ql/CashFlows/parcoupon.hpp, 870
- ql/CashFlows/shortfloatingcoupon.hpp, 871
- ql/CashFlows/shortindexedcoupon.hpp, 872
- ql/CashFlows/simplecashflow.hpp, 873
- ql/CashFlows/timebasket.hpp, 874
- ql/CashFlows/upfrontindexedcoupon.hpp, 875
- ql/Currencies/africa.hpp, 876
- ql/Currencies/america.hpp, 877
- ql/Currencies/asia.hpp, 879
- ql/Currencies/europe.hpp, 881
- ql/Currencies/exchangeratemanager.hpp, 884
- ql/Currencies/oceania.hpp, 885
- ql/currency.hpp, 886
- ql/dataformatters.hpp, 888
- ql/dataparsers.hpp, 889
- ql/date.hpp, 890
- ql/daycounter.hpp, 892
- ql/DayCounters/actual360.hpp, 893
- ql/DayCounters/actual365.hpp, 894
- ql/DayCounters/actual365fixed.hpp, 895
- ql/DayCounters/actualactual.hpp, 896
- ql/DayCounters/one.hpp, 897
- ql/DayCounters/simpliedaycounter.hpp, 898
- ql/DayCounters/thirty360.hpp, 899
- ql/discretizedasset.hpp, 900
- ql/disposable.hpp, 901
- ql/errors.hpp, 902
- ql/exchangerate.hpp, 904
- ql/exercise.hpp, 905
- ql/FiniteDifferences/americancondition.hpp, 906
- ql/FiniteDifferences/boundarycondition.hpp, 907
- ql/FiniteDifferences/bsmoperator.hpp, 908
- ql/FiniteDifferences/cranknicolson.hpp, 909
- ql/FiniteDifferences/dminus.hpp, 910
- ql/FiniteDifferences/dplus.hpp, 911
- ql/FiniteDifferences/dplusdminus.hpp, 912
- ql/FiniteDifferences/dzero.hpp, 913
- ql/FiniteDifferences/expliciteuler.hpp, 914
- ql/FiniteDifferences/fdtypedefs.hpp, 915
- ql/FiniteDifferences/finitedifferencemodel.hpp, 916
- ql/FiniteDifferences/impliciteuler.hpp, 917
- ql/FiniteDifferences/mixedscheme.hpp, 918
- ql/FiniteDifferences/onefactoroperator.hpp, 919
- ql/FiniteDifferences/shoutcondition.hpp, 920
- ql/FiniteDifferences/stepcondition.hpp, 921
- ql/FiniteDifferences/tridiagonaloperator.hpp, 922
- ql/FiniteDifferences/valueatcenter.hpp, 923
- ql/grid.hpp, 925
- ql/history.hpp, 926
- ql/index.hpp, 927
- ql/Indexes/audlibor.hpp, 928
- ql/Indexes/cadlibor.hpp, 929
- ql/Indexes/chflibor.hpp, 930
- ql/Indexes/euribor.hpp, 931
- ql/Indexes/gbpplibor.hpp, 932
- ql/Indexes/indexmanager.hpp, 933
- ql/Indexes/jpylibor.hpp, 934
- ql/Indexes/usdlibor.hpp, 935
- ql/Indexes/xibor.hpp, 936
- ql/Indexes/xibormanager.hpp, 937
- ql/Indexes/zarlibor.hpp, 938
- ql/instrument.hpp, 939
- ql/Instruments/asianoption.hpp, 940
- ql/Instruments/barrieroption.hpp, 941
- ql/Instruments/basketoption.hpp, 942
- ql/Instruments/bond.hpp, 943
- ql/Instruments/capfloor.hpp, 944
- ql/Instruments/cliquestoption.hpp, 945
- ql/Instruments/dividendvanillaoption.hpp, 946
- ql/Instruments/europeanoption.hpp, 947
- ql/Instruments/fixedcouponbond.hpp, 948
- ql/Instruments/forwardvanillaoption.hpp, 949
- ql/Instruments/multiassetoption.hpp, 950
- ql/Instruments/oneassetoption.hpp, 951

- ql/Instruments/oneassetstrikedoption.hpp, 952
- ql/Instruments/payoffs.hpp, 953
- ql/Instruments/quantoforwardvanillaoption.hpp, 955
- ql/Instruments/quantovanillaoption.hpp, 956
- ql/Instruments/simpleswap.hpp, 957
- ql/Instruments/stock.hpp, 958
- ql/Instruments/swap.hpp, 959
- ql/Instruments/swaption.hpp, 960
- ql/Instruments/vanillaoption.hpp, 961
- ql/interestrate.hpp, 962
- ql/Lattices/binomialtree.hpp, 963
- ql/Lattices/bsmlattice.hpp, 965
- ql/Lattices/lattice.hpp, 966
- ql/Lattices/lattice2d.hpp, 967
- ql/Lattices/tree.hpp, 968
- ql/Lattices/trinomialtree.hpp, 969
- ql/Math/array.hpp, 970
- ql/Math/beta.hpp, 971
- ql/Math/bicubicsplineinterpolation.hpp, 972
- ql/Math/bilinearinterpolation.hpp, 973
- ql/Math/binomialdistribution.hpp, 974
- ql/Math/bivariatenormaldistribution.hpp, 976
- ql/Math/chisquaredistribution.hpp, 977
- ql/Math/choleskydecomposition.hpp, 978
- ql/Math/comparison.hpp, 979
- ql/Math/cubicspline.hpp, 980
- ql/Math/discrepancystatistics.hpp, 981
- ql/Math/errorfunction.hpp, 982
- ql/Math/extrapolation.hpp, 983
- ql/Math/factorial.hpp, 984
- ql/Math/functional.hpp, 985
- ql/Math/gammadistribution.hpp, 986
- ql/Math/gaussianstatistics.hpp, 987
- ql/Math/generalstatistics.hpp, 988
- ql/Math/incompletegammahpp, 989
- ql/Math/incrementalstatistics.hpp, 990
- ql/Math/interpolation.hpp, 991
- ql/Math/interpolation2D.hpp, 992
- ql/Math/interpolationtraits.hpp, 993
- ql/Math/kronrodintegral.hpp, 994
- ql/Math/lexicographicalview.hpp, 995
- ql/Math/linearinterpolation.hpp, 996
- ql/Math/loglinearinterpolation.hpp, 997
- ql/Math/matrix.hpp, 998
- ql/Math/multicubicspline.hpp, 999
- ql/Math/normaldistribution.hpp, 1001
- ql/Math/poissondistribution.hpp, 1002
- ql/Math/primenumbers.hpp, 1003
- ql/Math/pseudosqrt.hpp, 1004
- ql/Math/riskstatistics.hpp, 1005
- ql/Math/rounding.hpp, 1007
- ql/Math/segmentintegral.hpp, 1008
- ql/Math/sequencestatistics.hpp, 1009
- ql/Math/simpsonintegral.hpp, 1011
- ql/Math/statistics.hpp, 1012
- ql/Math/svd.hpp, 1013
- ql/Math/symmetriceigenvalues.hpp, 1014
- ql/Math/symmetricschurdecomposition.hpp, 1015
- ql/Math/trapezoidintegral.hpp, 1016
- ql/money.hpp, 1017
- ql/MonteCarlo/brownianbridge.hpp, 1018
- ql/MonteCarlo/getcovariance.hpp, 1019
- ql/MonteCarlo/mctraits.hpp, 1020
- ql/MonteCarlo/mctypedefs.hpp, 1021
- ql/MonteCarlo/montecarlomodel.hpp, 1022
- ql/MonteCarlo/multipath.hpp, 1023
- ql/MonteCarlo/multipathgenerator.hpp, 1024
- ql/MonteCarlo/path.hpp, 1025
- ql/MonteCarlo/pathgenerator.hpp, 1026
- ql/MonteCarlo/pathpricer.hpp, 1027
- ql/MonteCarlo/sample.hpp, 1028
- ql/null.hpp, 1029
- ql/numericalmethod.hpp, 1030
- ql/Optimization/armijo.hpp, 1031
- ql/Optimization/conjugategradient.hpp, 1032
- ql/Optimization/constraint.hpp, 1033
- ql/Optimization/costfunction.hpp, 1034
- ql/Optimization/criteria.hpp, 1035
- ql/Optimization/leastsquare.hpp, 1036
- ql/Optimization/lineearch.hpp, 1037
- ql/Optimization/method.hpp, 1038
- ql/Optimization/problem.hpp, 1039
- ql/Optimization/simplex.hpp, 1040
- ql/Optimization/steepestdescent.hpp, 1041
- ql/option.hpp, 1042
- ql/Patterns/bridge.hpp, 1043
- ql/Patterns/composite.hpp, 1044
- ql/Patterns/curiouslyrecurring.hpp, 1045
- ql/Patterns/lazyobject.hpp, 1046
- ql/Patterns/observable.hpp, 1047
- ql/Patterns/singleton.hpp, 1048
- ql/Patterns/visitor.hpp, 1049
- ql/payoff.hpp, 1050
- ql/Pricers/discretegeometricapohpp, 1051
- ql/Pricers/discretegeometricasohpp, 1052
- ql/Pricers/fdamericanoption.hpp, 1053
- ql/Pricers/fdbermudanoption.hpp, 1054
- ql/Pricers/fdbsmoption.hpp, 1055
- ql/Pricers/fddividendamercanoption.hpp, 1056
- ql/Pricers/fddividendooption.hpp, 1057

- ql/Pricers/fddividendshoutoption.hpp, 1058
- ql/Pricers/fdeuropean.hpp, 1059
- ql/Pricers/fdmultiperiodoption.hpp, 1060
- ql/Pricers/fdshoutoption.hpp, 1061
- ql/Pricers/fdstepconditionoption.hpp, 1062
- ql/Pricers/mccliquestoption.hpp, 1063
- ql/Pricers/mcdiscretearithmeticapo.hpp, 1064
- ql/Pricers/mcdiscretearithmeticaso.hpp, 1065
- ql/Pricers/mceverest.hpp, 1066
- ql/Pricers/mchimalaya.hpp, 1067
- ql/Pricers/mcmaxbasket.hpp, 1068
- ql/Pricers/mcpagoda.hpp, 1069
- ql/Pricers/mcperformanceoption.hpp, 1070
- ql/Pricers/mcpricer.hpp, 1071
- ql/Pricers/singleassetoption.hpp, 1072
- ql/pricingengine.hpp, 1073
- ql/PricingEngines/americanpayoffatexpiry.hpp, 1074
- ql/PricingEngines/americanpayoffathit.hpp, 1075
- ql/PricingEngines/Asian/analytic_cont_geom_av_price.hpp, 1076
- ql/PricingEngines/Asian/analytic_discr_geom_av_price.hpp, 1077
- ql/PricingEngines/Asian/mc_discr_arith_av_price.hpp, 1078
- ql/PricingEngines/Asian/mc_discr_geom_av_price.hpp, 1079
- ql/PricingEngines/Asian/mcdiscreteasianengine.hpp, 1080
- ql/PricingEngines/Barrier/analyticbarrierengine.hpp, 1081
- ql/PricingEngines/Barrier/mcbarrierengine.hpp, 1082
- ql/PricingEngines/Basket/mcamericanbasketengine.hpp, 1083
- ql/PricingEngines/Basket/mcbasketengine.hpp, 1084
- ql/PricingEngines/Basket/stulzengine.hpp, 1085
- ql/PricingEngines/blackformula.hpp, 1086
- ql/PricingEngines/blackmodel.hpp, 1087
- ql/PricingEngines/CapFloor/analyticcapfloorengine.hpp, 1088
- ql/PricingEngines/CapFloor/blackcapfloorengine.hpp, 1089
- ql/PricingEngines/CapFloor/discretizedcapfloor.hpp, 1090
- ql/PricingEngines/CapFloor/treecapfloorengine.hpp, 1091
- ql/PricingEngines/Cliquet/analyticcliquestengine.hpp, 1092
- ql/PricingEngines/Cliquet/analyticperformanceengine.hpp, 1093
- ql/PricingEngines/Cliquet/mccliquestengine.hpp, 1094
- ql/PricingEngines/Forward/forwardengine.hpp, 1095
- ql/PricingEngines/Forward/forwardperformanceengine.hpp, 1096
- ql/PricingEngines/genericmodelengine.hpp, 1097
- ql/PricingEngines/latticeshortratemodelengine.hpp, 1098
- ql/PricingEngines/mcsimulation.hpp, 1099
- ql/PricingEngines/Quanto/quantoengine.hpp, 1100
- ql/PricingEngines/Swapoption/blackswapoptionengine.hpp, 1101
- ql/PricingEngines/Swapoption/discretizedswapoption.hpp, 1102
- ql/PricingEngines/Swapoption/g2swapoptionengine.hpp, 1103
- ql/PricingEngines/Swapoption/jamshidianswapoptionengine.hpp, 1104
- ql/PricingEngines/Swapoption/treeswapoptionengine.hpp, 1105
- ql/PricingEngines/Vanilla/analyticdigitalamericanengine.hpp, 1106
- ql/PricingEngines/Vanilla/analyticdividendeuropeanengine.hpp, 1107
- ql/PricingEngines/Vanilla/analyticeuropeanengine.hpp, 1108
- ql/PricingEngines/Vanilla/baroneadesiwhaleyengine.hpp, 1109
- ql/PricingEngines/Vanilla/binomialengine.hpp, 1110
- ql/PricingEngines/Vanilla/bjerkhundstenglandengine.hpp, 1111
- ql/PricingEngines/Vanilla/discretizedvanillaoption.hpp, 1112
- ql/PricingEngines/Vanilla/integralengine.hpp, 1113
- ql/PricingEngines/Vanilla/jumpdiffusionengine.hpp, 1114
- ql/PricingEngines/Vanilla/juquadraticengine.hpp, 1115
- ql/PricingEngines/Vanilla/mcdigitalengine.hpp, 1116
- ql/PricingEngines/Vanilla/mceuropeanengine.hpp, 1117
- ql/PricingEngines/Vanilla/mcvanillaengine.hpp, 1118
- ql/qldefines.hpp, 1119
- ql/quote.hpp, 1121

- ql/RandomNumbers/boxmullergaussianrng.hpp, 1122
- ql/RandomNumbers/centrallimitgaussianrng.hpp, 1123
- ql/RandomNumbers/faurersg.hpp, 1124
- ql/RandomNumbers/haltonrsg.hpp, 1125
- ql/RandomNumbers/inversecumgaussianrng.hpp, 1126
- ql/RandomNumbers/inversecumgaussianrsg.hpp, 1127
- ql/RandomNumbers/inversecumulativerng.hpp, 1128
- ql/RandomNumbers/inversecumulativersg.hpp, 1129
- ql/RandomNumbers/knuthuniformrng.hpp, 1130
- ql/RandomNumbers/lecuyeruniformrng.hpp, 1131
- ql/RandomNumbers/mt19937uniformrng.hpp, 1132
- ql/RandomNumbers/randomizedlds.hpp, 1133
- ql/RandomNumbers/randomsequencegenerator.hpp, 1134
- ql/RandomNumbers/rngtraits.hpp, 1135
- ql/RandomNumbers/rngtraits.hpp
 - PoissonPseudoRandom, 1136
 - PseudoRandom, 1136
- ql/RandomNumbers/seedgenerator.hpp, 1137
- ql/RandomNumbers/sobolrsg.hpp, 1138
- ql/relinkablehandle.hpp, 1139
- ql/schedule.hpp, 1140
- ql/settings.hpp, 1141
- ql/ShortRateModels/calibrationhelper.hpp, 1142
- ql/ShortRateModels/CalibrationHelpers/caphelper.hpp, 1143
- ql/ShortRateModels/CalibrationHelpers/swaptionhelper.hpp, 1144
- ql/ShortRateModels/model.hpp, 1145
- ql/ShortRateModels/onefactormodel.hpp, 1146
- ql/ShortRateModels/OneFactorModels/blackkarasinski.hpp, 1147
- ql/ShortRateModels/OneFactorModels/coxingersollrsg.hpp, 1148
- ql/ShortRateModels/OneFactorModels/extendedcoxingersollrsg.hpp, 1149
- ql/ShortRateModels/OneFactorModels/hullwhite.hpp, 1150
- ql/ShortRateModels/OneFactorModels/vasicek.hpp, 1151
- ql/ShortRateModels/parameter.hpp, 1152
- ql/ShortRateModels/twofactormodel.hpp, 1153
- ql/ShortRateModels/TwoFactorModels/g2.hpp, 1154
- ql/solver1d.hpp, 1155
- ql/Solvers1D/bisection.hpp, 1156
- ql/Solvers1D/brent.hpp, 1157
- ql/Solvers1D/falseposition.hpp, 1158
- ql/Solvers1D/newton.hpp, 1159
- ql/Solvers1D/newtonsafe.hpp, 1160
- ql/Solvers1D/ridder.hpp, 1161
- ql/Solvers1D/secant.hpp, 1162
- ql/stochasticprocess.hpp, 1163
- ql/swaptionvolstructure.hpp, 1164
- ql/termstructure.hpp, 1165
- ql/TermStructures/affinetermstructure.hpp, 1166
- ql/TermStructures/compoundforward.hpp, 1167
- ql/TermStructures/discountcurve.hpp, 1168
- ql/TermStructures/discountstructure.hpp, 1169
- ql/TermStructures/drifttermstructure.hpp, 1170
- ql/TermStructures/extendeddiscountcurve.hpp, 1171
- ql/TermStructures/flatforward.hpp, 1172
- ql/TermStructures/forwardspreadetermstructure.hpp, 1173
- ql/TermStructures/forwardstructure.hpp, 1174
- ql/TermStructures/impliedtermstructure.hpp, 1175
- ql/TermStructures/piecewiseflatforward.hpp, 1176
- ql/TermStructures/quantotermstructure.hpp, 1177
- ql/TermStructures/ratehelpers.hpp, 1178
- ql/TermStructures/zerocurve.hpp, 1179
- ql/TermStructures/zerospreadetermstructure.hpp, 1180
- ql/TermStructures/zeroyieldstructure.hpp, 1181
- ql/types.hpp, 1182
- ql/Utilities/combiningiterator.hpp, 1184
- ql/Utilities/couplingiterator.hpp, 1185
- ql/Utilities/filteringiterator.hpp, 1186
- ql/Utilities/iterable.hpp, 1187
- ql/Utilities/processingiterator.hpp, 1188
- ql/Utilities/steppingiterator.hpp, 1189
- ql/Volatilities/blackconstantvol.hpp, 1190
- ql/Volatilities/blackvariancecurve.hpp, 1191
- ql/Volatilities/blackvariancesurface.hpp, 1192

ql/Volatilities/capflatvolvector.hpp, 1193
 ql/Volatilities/capletconstantvol.hpp, 1194
 ql/Volatilities/IMPLIEDVOLTERMSTRUCTURE.hpp, 1195
 ql/Volatilities/localconstantvol.hpp, 1196
 ql/Volatilities/localvolcurve.hpp, 1197
 ql/Volatilities/localvolsurface.hpp, 1198
 ql/Volatilities/swaptionvolmatrix.hpp, 1199
 ql/voltermstructure.hpp, 1200
 QL_ASSERT
 errors.hpp, 903
 QL_DUMMY_RETURN
 miscMacros, 124
 QL_ENSURE
 errors.hpp, 903
 QL_EPSILON
 limitMacros, 126
 QL_FULL_ITERATOR_SUPPORT
 iteratorMacros, 132
 QL_IO_INIT
 miscMacros, 124
 QL_MAX_INTEGER
 limitMacros, 126
 QL_MAX_REAL
 limitMacros, 126
 QL_MIN_INTEGER
 limitMacros, 126
 QL_MIN_POSITIVE_REAL
 limitMacros, 126
 QL_MIN_REAL
 limitMacros, 126
 QL_REQUIRE
 errors.hpp, 903
 QL_REVERSE_ITERATOR
 iteratorMacros, 132
 QL_SPECIALIZE_ITERATOR_TRAITS
 iteratorMacros, 132
 QL_TYPENAME
 templateMacros, 131
 QuantLib::Actual360, 133
 QuantLib::Actual365Fixed, 134
 QuantLib::ActualActual, 135
 QuantLib::AcyclicVisitor, 136
 QuantLib::AdditiveEQPBinomialTree, 137
 QuantLib::AffineModel, 138
 QuantLib::AffineTermStructure, 139
 QuantLib::AffineTermStructure
 AffineTermStructure, 140
 update, 140
 QuantLib::AmericanCondition, 141
 QuantLib::AmericanCondition
 applyTo, 141
 QuantLib::AmericanExercise, 142

QuantLib::AmericanPayoffAtExpiry, 143
 QuantLib::AmericanPayoffAtHit, 144
 QuantLib::AnalyticBarrierEngine, 145
 QuantLib::AnalyticCapFloorEngine, 146
 QuantLib::AnalyticCliquetEngine, 147
 QuantLib::AnalyticContinuousGeometricAveragePriceAsianEngine, 148
 QuantLib::AnalyticDigitalAmericanEngine, 149
 QuantLib::AnalyticDiscreteGeometricAveragePriceAsianEngine, 150
 QuantLib::AnalyticDividendEuropeanEngine, 151
 QuantLib::AnalyticEuropeanEngine, 152
 QuantLib::AnalyticPerformanceEngine, 153
 QuantLib::Arguments, 154
 QuantLib::ArmijoLineSearch, 155
 QuantLib::Array, 156
 QuantLib::Array
 operator<<, 158
 QuantLib::ArrayFormatter, 159
 QuantLib::ARSCurrency, 160
 QuantLib::AssetOrNothingPayoff, 161
 QuantLib::ATSCurrency, 162
 QuantLib::AUDCurrency, 163
 QuantLib::AUDLiber, 164
 QuantLib::Average, 165
 QuantLib::BaroneAdesiWhaleyApproximationEngine, 166
 QuantLib::Barrier, 167
 QuantLib::BarrierOption, 168
 QuantLib::BarrierOption
 performCalculations, 169
 setupArguments, 169
 QuantLib::BarrierOption::arguments, 170
 QuantLib::BarrierOption::engine, 171
 QuantLib::BaseTermStructure, 172
 QuantLib::BaseTermStructure
 BaseTermStructure, 173
 todaysDate, 173
 update, 173
 QuantLib::BasketOption, 175
 QuantLib::BasketOption
 setupArguments, 175
 QuantLib::BasketOption::arguments, 176
 QuantLib::BasketOption::engine, 177
 QuantLib::BDTCurrency, 178
 QuantLib::BEFCurrency, 179
 QuantLib::Beijing, 180
 QuantLib::BermudanExercise, 181
 QuantLib::BGLCurrency, 182
 QuantLib::BicubicSpline, 183
 QuantLib::BicubicSpline
 BicubicSpline, 183

- QuantLib::BilinearInterpolation, 184
- QuantLib::BilinearInterpolation
 - BilinearInterpolation, 184
- QuantLib::BinomialDistribution, 185
- QuantLib::BinomialTree, 186
- QuantLib::BinomialVanillaEngine, 187
- QuantLib::Bisection, 188
- QuantLib::BivariateCumulativeNormalDistribution, 189
- QuantLib::Bjerk SundStenslandApproximationEngine, 190
- QuantLib::BlackCapFloorEngine, 191
- QuantLib::BlackConstantVol, 192
- QuantLib::BlackFormula, 194
- QuantLib::BlackFormula
 - itmAssetProbability, 194
 - itmCashProbability, 194
- QuantLib::BlackKarasinski, 195
- QuantLib::BlackKarasinskiDynamics, 196
- QuantLib::BlackModel, 197
- QuantLib::BlackModel
 - formula, 197
 - itmProbability, 198
 - update, 197
- QuantLib::BlackScholesLattice, 199
- QuantLib::BlackScholesProcess, 200
- QuantLib::BlackScholesProcess
 - diffusion, 201
 - drift, 201
 - evolve, 201
 - update, 201
- QuantLib::BlackSwaptionEngine, 202
- QuantLib::BlackVarianceCurve, 203
- QuantLib::BlackVarianceSurface, 205
- QuantLib::BlackVarianceTermStructure, 207
- QuantLib::BlackVarianceTermStructure
 - BlackVarianceTermStructure, 208
 - blackVolImpl, 208
- QuantLib::BlackVolatilityTermStructure, 209
- QuantLib::BlackVolatilityTermStructure
 - blackVarianceImpl, 210
 - BlackVolatilityTermStructure, 210
- QuantLib::BlackVolTermStructure, 211
- QuantLib::BlackVolTermStructure
 - BlackVolTermStructure, 213
- QuantLib::Bond, 214
- QuantLib::Bond
 - accruedAmount, 215
 - cleanPrice, 215
 - dirtyPrice, 215
 - performCalculations, 215
 - yield, 215
- QuantLib::BoundaryCondition, 217
- QuantLib::BoundaryCondition
 - applyAfterApplying, 217
 - applyAfterSolving, 217
 - applyBeforeApplying, 217
 - applyBeforeSolving, 217
 - setTime, 218
 - Side, 217
- QuantLib::BoundaryConstraint, 219
- QuantLib::BoxMullerGaussianRng, 220
- QuantLib::BPSBasketCalculator, 221
- QuantLib::BPSCalculator, 222
- QuantLib::Brent, 223
- QuantLib::Bridge, 224
- QuantLib::BRLCurrency, 225
- QuantLib::BrownianBridge, 226
- QuantLib::BSMOperator, 227
- QuantLib::Budapest, 228
- QuantLib::BYRCurrency, 229
- QuantLib::CADCurrency, 230
- QuantLib::CADLibor, 231
- QuantLib::Calendar, 232
- QuantLib::Calendar
 - addHoliday, 234
 - adjust, 234
 - advance, 234, 235
 - Calendar, 233
 - isBusinessDay, 234
 - isEndOfMonth, 234
 - isHoliday, 234
 - name, 234
 - operator==, 235
 - removeHoliday, 234
- QuantLib::Calendar::WesternImpl, 236
- QuantLib::CalendarImpl, 237
- QuantLib::CalibrationHelper, 238
- QuantLib::CalibrationHelper
 - update, 239
- QuantLib::Cap, 240
- QuantLib::CapFloor, 241
- QuantLib::CapFloor
 - setupArguments, 242
- QuantLib::CapFloor::arguments, 243
- QuantLib::CapFloor::results, 244
- QuantLib::CapletConstantVolatility, 245
- QuantLib::CapletVolatilityStructure, 246
- QuantLib::CapletVolatilityStructure
 - CapletVolatilityStructure, 247
- QuantLib::CapVolatilityStructure, 248
- QuantLib::CapVolatilityStructure
 - CapVolatilityStructure, 249
- QuantLib::CapVolatilityVector, 250
- QuantLib::CapVolatilityVector
 - CapVolatilityVector, 250
 - update, 251
- QuantLib::CashFlow, 252

- QuantLib::CashFlow
 - amount, 252
- QuantLib::CashOrNothingPayoff, 253
- QuantLib::CeilingTruncation, 254
- QuantLib::CHFCurrency, 255
- QuantLib::CHFLibor, 256
- QuantLib::CLGaussianRng, 257
- QuantLib::CliquetOption, 258
- QuantLib::CliquetOption
 - setupArguments, 259
- QuantLib::CliquetOption::arguments, 260
- QuantLib::CliquetOption::engine, 261
- QuantLib::ClosestRounding, 262
- QuantLib::CLPCurrency, 263
- QuantLib::CNYCurrency, 264
- QuantLib::Collar, 265
- QuantLib::combining_iterator, 266
- QuantLib::Composite, 268
- QuantLib::CompositeConstraint, 269
- QuantLib::CompositeQuote, 270
- QuantLib::CompositeQuote
 - update, 270
- QuantLib::CompoundForward, 271
- QuantLib::CompoundForward
 - CompoundForward, 272
 - compoundForwardImpl, 272
 - discountImpl, 272
 - zeroYieldImpl, 272
- QuantLib::CompoundingRuleFormatter, 274
- QuantLib::ConjugateGradient, 275
- QuantLib::ConstantParameter, 276
- QuantLib::Constraint, 277
- QuantLib::ConstraintImpl, 278
- QuantLib::ContinuousAveragingAsianOption, 279
- QuantLib::ContinuousAveragingAsian-Option
 - setupArguments, 279
- QuantLib::ContinuousAveragingAsianOption::arguments, 281
- QuantLib::ContinuousAveragingAsianOption::engine, 282
- QuantLib::COPCurrency, 283
- QuantLib::Copenhagen, 284
- QuantLib::CostFunction, 285
- QuantLib::coupling_iterator, 286
- QuantLib::Coupon, 288
- QuantLib::Coupon
 - Coupon, 289
- QuantLib::CovarianceDecomposition, 290
- QuantLib::CovarianceDecomposition
 - correlationMatrix, 290
 - CovarianceDecomposition, 290
 - standardDeviations, 290
 - variances, 290
- QuantLib::CoxIngersollRoss, 291
- QuantLib::CoxIngersollRoss::Dynamics, 293
- QuantLib::CoxRossRubinstein, 294
- QuantLib::CrankNicolson, 295
- QuantLib::Cubic, 296
- QuantLib::CubicSpline, 297
 - FirstDerivative, 298
 - Lagrange, 298
 - NotAKnot, 298
 - Periodic, 298
 - SecondDerivative, 298
- QuantLib::CubicSpline
 - BoundaryCondition, 298
 - CubicSpline, 298
- QuantLib::CumulativeBinomialDistribution, 299
- QuantLib::CumulativeNormalDistribution, 300
- QuantLib::CumulativePoissonDistribution, 301
- QuantLib::CuriouslyRecurringTemplate, 302
- QuantLib::Currency, 303
- QuantLib::Currency
 - Currency, 306
 - format, 306
- QuantLib::CurrencyFormatter, 307
- QuantLib::CYPCurrency, 308
- QuantLib::CZKCurrency, 309
- QuantLib::Date, 310
- QuantLib::Date
 - isEndOfMonth, 312
 - lastDayOfMonth, 312
 - nextIMMdate, 313
 - nextWeekday, 313
 - nthWeekday, 313
 - operator<<, 314
 - plus, 313
 - plusDays, 312
 - plusMonths, 313
 - plusWeeks, 313
 - plusYears, 313
- QuantLib::DateFormatter, 315
- QuantLib::DayCounter, 316
- QuantLib::DayCounter
 - DayCounter, 317
 - name, 317
 - operator==, 317
- QuantLib::DayCounterImpl, 318
- QuantLib::DecimalFormatter, 319
- QuantLib::DEMCurrency, 320
- QuantLib::DepositRateHelper, 321

- QuantLib::DepositRateHelper
 - latestDate, [321](#)
 - setTermStructure, [321](#)
- QuantLib::DerivedQuote, [323](#)
- QuantLib::DerivedQuote
 - update, [323](#)
- QuantLib::DirichletBC, [324](#)
- QuantLib::DirichletBC
 - setTime, [324](#)
- QuantLib::DiscountCurve, [325](#)
- QuantLib::DiscountCurve
 - DiscountCurve, [326](#)
- QuantLib::DiscountStructure, [327](#)
- QuantLib::DiscountStructure
 - compoundForwardImpl, [328](#)
 - forwardImpl, [328](#)
 - zeroYieldImpl, [328](#)
- QuantLib::DiscrepancyStatistics, [329](#)
- QuantLib::DiscreteAveragingAsianOption, [330](#)
- QuantLib::DiscreteAveragingAsianOption
 - setupArguments, [331](#)
- QuantLib::DiscreteAveragingAsianOption::arguments, [332](#)
- QuantLib::DiscreteAveragingAsianOption::engine, [333](#)
- QuantLib::DiscreteGeometricAPO, [334](#)
- QuantLib::DiscreteGeometricASO, [335](#)
- QuantLib::DiscretizedAsset, [336](#)
- QuantLib::DiscretizedAsset
 - addTimesTo, [338](#)
 - adjustValues, [337](#)
 - DiscretizedAsset, [337](#)
 - isOnTime, [338](#)
 - mandatoryTimes, [337](#)
 - postAdjustValues, [337](#)
 - postAdjustValuesImpl, [338](#)
 - preAdjustValues, [337](#)
 - preAdjustValuesImpl, [338](#)
 - reset, [337](#)
- QuantLib::DiscretizedDiscountBond, [339](#)
- QuantLib::DiscretizedDiscountBond
 - DiscretizedDiscountBond, [339](#)
 - reset, [339](#)
- QuantLib::DiscretizedOption, [340](#)
- QuantLib::DiscretizedOption
 - mandatoryTimes, [340](#)
 - postAdjustValuesImpl, [341](#)
 - reset, [340](#)
- QuantLib::Disposable, [342](#)
- QuantLib::DividendVanillaOption, [343](#)
- QuantLib::DividendVanillaOption
 - setupArguments, [343](#)
- QuantLib::DividendVanillaOption::arguments, [345](#)
- QuantLib::DividendVanillaOption::engine, [346](#)
- QuantLib::DKKCurrency, [347](#)
- QuantLib::DMinus, [348](#)
- QuantLib::DownRounding, [349](#)
- QuantLib::DPlus, [350](#)
- QuantLib::DPlusDMinus, [351](#)
- QuantLib::DriftTermStructure, [352](#)
- QuantLib::DriftTermStructure
 - today'sDate, [353](#)
- QuantLib::DZero, [354](#)
- QuantLib::EarlyExercise, [355](#)
- QuantLib::EEKCurrency, [356](#)
- QuantLib::EndCriteria, [357](#)
- QuantLib::EqualJumpsBinomialTree, [359](#)
- QuantLib::EqualProbabilitiesBinomialTree, [360](#)
- QuantLib::Error, [361](#)
- QuantLib::Error
 - ~Error, [361](#)
 - Error, [361](#)
- QuantLib::ErrorFunction, [362](#)
- QuantLib::ESPCurrency, [363](#)
- QuantLib::EulerDiscretization, [364](#)
- QuantLib::EulerDiscretization
 - expectation, [364](#)
 - variance, [364](#)
- QuantLib::EURCurrency, [365](#)
- QuantLib::Euribor, [366](#)
- QuantLib::EuroFormatter, [367](#)
- QuantLib::EuropeanExercise, [368](#)
- QuantLib::EuropeanOption, [369](#)
- QuantLib::ExchangeRate, [370](#)
 - Derived, [371](#)
 - Direct, [371](#)
- QuantLib::ExchangeRate
 - ExchangeRate, [371](#)
 - Type, [371](#)
- QuantLib::ExchangeRateManager, [372](#)
- QuantLib::ExchangeRateManager
 - add, [372](#)
 - lookup, [372](#)
- QuantLib::Exercise, [374](#)
- QuantLib::ExplicitEuler, [375](#)
- QuantLib::ExtendedCoxIngersollRoss, [376](#)
- QuantLib::ExtendedCoxIngersollRoss::Dynamics, [377](#)
- QuantLib::ExtendedCoxIngersollRoss::FittingParameter, [378](#)
- QuantLib::ExtendedDiscountCurve, [379](#)
- QuantLib::ExtendedDiscountCurve
 - compoundForwardImpl, [380](#)

- ExtendedDiscountCurve, 380
- update, 380
- QuantLib::Extrapolator, 381
- QuantLib::Factorial, 382
- QuantLib::FalsePosition, 383
- QuantLib::FaureRsg, 384
- QuantLib::FdAmericanOption, 385
- QuantLib::FdBermudanOption, 386
- QuantLib::FdBsmOption, 387
- QuantLib::FdDividendAmericanOption, 389
- QuantLib::FdDividendShoutOption, 390
- QuantLib::FdEuropean, 391
- QuantLib::FdStepConditionOption, 392
- QuantLib::filtering_iterator, 393
- QuantLib::FIMCurrency, 394
- QuantLib::FiniteDifferenceModel, 395
- QuantLib::FiniteDifferenceModel
 - rollback, 395
- QuantLib::FixedCouponBond, 396
- QuantLib::FixedRateCoupon, 397
- QuantLib::FixedRateCoupon
 - amount, 398
- QuantLib::FlatForward, 399
- QuantLib::FlatForward
 - FlatForward, 400
- QuantLib::FloatingRateCoupon, 401
- QuantLib::FloatingRateCoupon
 - fixing, 402
- QuantLib::Floor, 403
- QuantLib::FloorTruncation, 404
- QuantLib::ForwardEngine, 405
- QuantLib::ForwardOptionArguments, 406
- QuantLib::ForwardPerformanceEngine, 407
- QuantLib::ForwardRateStructure, 408
- QuantLib::ForwardRateStructure
 - compoundForwardImpl, 409
 - discountImpl, 409
 - ForwardRateStructure, 409
 - zeroYieldImpl, 409
- QuantLib::ForwardSpreadedTermStructure, 410
- QuantLib::ForwardSpreadedTermStructure
 - todayDate, 411
 - zeroYieldImpl, 411
- QuantLib::ForwardVanillaOption, 412
- QuantLib::ForwardVanillaOption
 - performCalculations, 413
 - setupArguments, 413
- QuantLib::FraRateHelper, 414
- QuantLib::FraRateHelper
 - latestDate, 415
 - setTermStructure, 414
- QuantLib::FrequencyFormatter, 416
- QuantLib::FRFCurrency, 417
- QuantLib::FuturesRateHelper, 418
- QuantLib::FuturesRateHelper
 - latestDate, 418
- QuantLib::G2, 419
- QuantLib::G2::FittingParameter, 421
- QuantLib::G2SwaptionEngine, 422
- QuantLib::GammaFunction, 423
- QuantLib::GapPayoff, 424
- QuantLib::GaussianStatistics, 425
- QuantLib::GaussianStatistics
 - gaussianDownsideDeviation, 425
 - gaussianDownsideVariance, 425
 - gaussianExpectedShortfall, 426
 - gaussianPercentile, 426
 - gaussianPotentialUpside, 426
 - gaussianRegret, 426
 - gaussianTopPercentile, 426
 - gaussianValueAtRisk, 426
- QuantLib::GBPCurrency, 428
- QuantLib::GBPLibor, 429
- QuantLib::GeneralStatistics, 430
- QuantLib::GeneralStatistics
 - add, 432
 - errorEstimate, 431
 - expectationValue, 432
 - kurtosis, 431
 - max, 432
 - mean, 431
 - min, 431
 - percentile, 432
 - skewness, 431
 - standardDeviation, 431
 - topPercentile, 432
 - variance, 431
- QuantLib::GenericEngine, 433
- QuantLib::GenericModelEngine, 434
- QuantLib::GenericModelEngine
 - update, 434
- QuantLib::GenericRiskStatistics, 435
- QuantLib::GenericRiskStatistics
 - averageShortfall, 437
 - downsideDeviation, 436
 - downsideVariance, 435
 - expectedShortfall, 436
 - potentialUpside, 436
 - regret, 436
 - semiDeviation, 435
 - semiVariance, 435
 - shortfall, 436
 - valueAtRisk, 436
- QuantLib::GeometricBrownianMotionProcess, 438
- QuantLib::Germany, 439

- Eurex, [441](#)
- FrankfurtStockExchange, [441](#)
- Settlement, [441](#)
- Xetra, [441](#)
- QuantLib::Germany
 - Market, [441](#)
- QuantLib::GRDCurrency, [442](#)
- QuantLib::Greeks, [443](#)
- QuantLib::HaltonRsg, [444](#)
- QuantLib::Handle, [445](#)
- QuantLib::Handle
 - Handle, [445](#)
 - isNull, [445](#)
 - linkTo, [445](#)
- QuantLib::Helsinki, [447](#)
- QuantLib::History, [448](#)
- QuantLib::History
 - History, [449](#)
- QuantLib::History::const_iterator, [451](#)
- QuantLib::History::Entry, [452](#)
- QuantLib::HKDCurrency, [453](#)
- QuantLib::HongKong, [454](#)
- QuantLib::HUFCurrency, [455](#)
- QuantLib::HullWhite, [456](#)
- QuantLib::HullWhite::Dynamics, [457](#)
- QuantLib::HullWhite::FittingParameter, [458](#)
- QuantLib::ICGaussianRng, [459](#)
- QuantLib::ICGaussianRsg, [460](#)
- QuantLib::IEPCurrency, [461](#)
- QuantLib::ILSCurrency, [462](#)
- QuantLib::ImplicitEuler, [463](#)
- QuantLib::ImpliedTermStructure, [464](#)
- QuantLib::ImpliedTermStructure
 - ImpliedTermStructure, [465](#)
- QuantLib::ImpliedVolTermStructure, [466](#)
- QuantLib::InArrearIndexedCoupon, [468](#)
- QuantLib::IncrementalStatistics, [470](#)
- QuantLib::IncrementalStatistics
 - add, [472](#)
 - addSequence, [472](#)
 - downsideDeviation, [471](#)
 - downsideVariance, [471](#)
 - errorEstimate, [471](#)
 - kurtosis, [472](#)
 - max, [472](#)
 - mean, [471](#)
 - min, [472](#)
 - skewness, [472](#)
 - standardDeviation, [471](#)
 - variance, [471](#)
- QuantLib::Index, [473](#)
- QuantLib::Index
 - fixing, [473](#)
 - name, [473](#)
- QuantLib::IndexedCoupon, [474](#)
- QuantLib::IndexedCoupon
 - amount, [475](#)
 - fixing, [475](#)
 - update, [475](#)
- QuantLib::IndexManager, [476](#)
- QuantLib::INRCurrency, [477](#)
- QuantLib::Instrument, [478](#)
- QuantLib::Instrument
 - calculate, [479](#)
 - performCalculations, [479](#)
 - setPricingEngine, [479](#)
 - setupArguments, [479](#)
 - setupExpired, [479](#)
- QuantLib::IntegerFormatter, [481](#)
- QuantLib::IntegralEngine, [482](#)
- QuantLib::InterestRate, [483](#)
- QuantLib::InterestRate
 - compoundFactor, [484](#)
 - discountFactor, [484](#)
 - equivalentRate, [485](#)
 - impliedRate, [484](#), [485](#)
- QuantLib::InterestRateFormatter, [486](#)
- QuantLib::Interpolation, [487](#)
- QuantLib::Interpolation2D, [488](#)
- QuantLib::Interpolation2D::templateImpl, [489](#)
- QuantLib::Interpolation2DImpl, [490](#)
- QuantLib::Interpolation::templateImpl, [491](#)
- QuantLib::InterpolationImpl, [492](#)
- QuantLib::InverseCumulativeNormal, [493](#)
- QuantLib::InverseCumulativePoisson, [494](#)
- QuantLib::InverseCumulativeRng, [495](#)
- QuantLib::InverseCumulativeRsg, [496](#)
- QuantLib::IQDCurrency, [497](#)
- QuantLib::IRRCurrency, [498](#)
- QuantLib::ISKCurrency, [499](#)
- QuantLib::Italy, [500](#)
 - Exchange, [501](#)
 - Settlement, [501](#)
- QuantLib::Italy
 - Market, [501](#)
- QuantLib::ITLCurrency, [502](#)
- QuantLib::JamshidianSwaptionEngine, [503](#)
- QuantLib::JarrowRudd, [504](#)
- QuantLib::Johannesburg, [505](#)
- QuantLib::JointCalendar, [506](#)
- QuantLib::JPYCurrency, [507](#)
- QuantLib::JPYLibor, [508](#)
- QuantLib::JumpDiffusionEngine, [509](#)
- QuantLib::JuQuadraticApproximationEngine, [510](#)
- QuantLib::KnuthUniformRng, [511](#)
- QuantLib::KnuthUniformRng

- KnuthUniformRng, 511
- next, 511
- QuantLib::KronrodIntegral, 512
- QuantLib::KRWCurrency, 513
- QuantLib::KWDCurrency, 514
- QuantLib::Lattice, 515
- QuantLib::Lattice
 - partialRollback, 516
 - rollback, 516
- QuantLib::Lattice2D, 517
- QuantLib::LatticeShortRateModelEngine, 518
- QuantLib::LatticeShortRateModelEngine
 - update, 518
- QuantLib::LazyObject, 519
- QuantLib::LazyObject
 - calculate, 520
 - freeze, 520
 - performCalculations, 520
 - recalculate, 520
 - unfreeze, 520
 - update, 519
- QuantLib::LeastSquareFunction, 521
- QuantLib::LeastSquareProblem, 522
- QuantLib::LeastSquareProblem
 - targetValueAndGradient, 522
- QuantLib::LecuyerUniformRng, 523
- QuantLib::LecuyerUniformRng
 - LecuyerUniformRng, 523
 - next, 523
- QuantLib::LeisenReimer, 524
- QuantLib::LexicographicalView, 525
- QuantLib::Linear, 527
- QuantLib::LinearInterpolation, 528
- QuantLib::LinearInterpolation
 - LinearInterpolation, 528
- QuantLib::LineSearch, 529
- QuantLib::Link, 531
- QuantLib::Link
 - isNull, 532
 - Link, 531
 - linkTo, 532
- QuantLib::LocalConstantVol, 533
- QuantLib::LocalVolCurve, 534
- QuantLib::LocalVolCurve
 - localVolImpl, 535
- QuantLib::LocalVolSurface, 536
- QuantLib::LocalVolTermStructure, 538
- QuantLib::LocalVolTermStructure
 - LocalVolTermStructure, 539
- QuantLib::LogLinear, 540
- QuantLib::LogLinearInterpolation, 541
- QuantLib::LogLinearInterpolation
 - LogLinearInterpolation, 541
- QuantLib::lowest_category_iterator, 542
- QuantLib::LTLCurrency, 543
- QuantLib::LUTCurrency, 544
- QuantLib::LVLCurrency, 545
- QuantLib::MakeSchedule, 546
- QuantLib::Matrix, 547
- QuantLib::Matrix
 - operator+=, 549
 - operator<<, 549
 - pseudoSqrt, 549
 - rankReducedSqrt, 550
- QuantLib::MatrixFormatter, 551
- QuantLib::MCAmericanBasketEngine, 552
- QuantLib::MCBarrierEngine, 553
- QuantLib::MCBasketEngine, 555
- QuantLib::McCliquetOption, 557
- QuantLib::MCDigitalEngine, 558
- QuantLib::MCDiscreteArithmeticAPEngine, 560
- QuantLib::MCDiscreteArithmeticAPO, 562
- QuantLib::MCDiscreteArithmeticASO, 563
- QuantLib::MCDiscreteAveragingAsianEngine, 564
- QuantLib::MCDiscreteGeometricAPEngine, 566
- QuantLib::MCEuropeanEngine, 567
- QuantLib::McEverest, 568
- QuantLib::McHimalaya, 569
- QuantLib::McMaxBasket, 570
- QuantLib::McPagoda, 571
- QuantLib::McPerformanceOption, 572
- QuantLib::McPricer, 573
- QuantLib::McSimulation, 574
- QuantLib::MCVanillaEngine, 576
- QuantLib::MersenneTwisterUniformRng, 578
- QuantLib::MersenneTwisterUniformRng
 - MersenneTwisterUniformRng, 578
 - next, 578
- QuantLib::Merton76Process, 579
- QuantLib::Merton76Process
 - diffusion, 579
 - drift, 579
 - evolve, 580
- QuantLib::MixedScheme, 581
- QuantLib::Money, 583
 - AutomatedConversion, 584
 - BaseCurrencyConversion, 584
 - NoConversion, 584
- QuantLib::Money
 - ConversionType, 584
- QuantLib::MoneyFormatter, 585
- QuantLib::MonotonicCubicSpline, 586
- QuantLib::MonotonicCubicSpline

- MonotonicCubicSpline, 586
- QuantLib::MonteCarloModel, 587
- QuantLib::MoreGreeks, 588
- QuantLib::MoroInverseCumulativeNormal, 589
- QuantLib::MTLCurrency, 590
- QuantLib::MultiAssetOption, 591
- QuantLib::MultiAssetOption
 - performCalculations, 592
 - setupArguments, 592
 - setupExpired, 592
- QuantLib::MultiAssetOption::arguments, 593
- QuantLib::MultiAssetOption::results, 594
- QuantLib::MultiCubicSpline, 595
- QuantLib::MultiPath, 596
- QuantLib::MultiPathGenerator, 597
- QuantLib::MXNCurrency, 598
- QuantLib::NaturalCubicSpline, 599
- QuantLib::NaturalCubicSpline
 - NaturalCubicSpline, 599
- QuantLib::NaturalMonotonicCubicSpline, 600
- QuantLib::NaturalMonotonicCubicSpline
 - NaturalMonotonicCubicSpline, 600
- QuantLib::NeumannBC, 601
- QuantLib::NeumannBC
 - setTime, 601
- QuantLib::Newton, 602
- QuantLib::NewtonSafe, 603
- QuantLib::NLGCurrency, 604
- QuantLib::NoConstraint, 605
- QuantLib::NOKCurrency, 606
- QuantLib::NonLinearLeastSquare, 607
- QuantLib::NormalDistribution, 608
- QuantLib::NPRCurrency, 609
- QuantLib::Null, 610
- QuantLib::NullCalendar, 611
- QuantLib::NullParameter, 612
- QuantLib::NumericalMethod, 613
- QuantLib::NumericalMethod
 - partialRollback, 613
 - rollback, 613
- QuantLib::NZDCurrency, 615
- QuantLib::Observable, 616
- QuantLib::Observable
 - notifyObservers, 616
- QuantLib::Observer, 617
- QuantLib::Observer
 - update, 618
- QuantLib::OneAssetOption, 619
- QuantLib::OneAssetOption
 - impliedVolatility, 620
 - performCalculations, 620
 - setupArguments, 620
 - setupExpired, 620
- QuantLib::OneAssetOption::arguments, 622
- QuantLib::OneAssetOption::results, 623
- QuantLib::OneAssetStrikedOption, 624
- QuantLib::OneAssetStrikedOption
 - performCalculations, 624
 - setupArguments, 624
- QuantLib::OneDayCounter, 626
- QuantLib::OneFactorAffineModel, 627
- QuantLib::OneFactorModel, 628
- QuantLib::OneFactorModel::ShortRateDynamics, 629
- QuantLib::OneFactorModel::ShortRateTree, 630
- QuantLib::OneFactorOperator, 631
- QuantLib::OptimizationMethod, 632
- QuantLib::Option, 634
- QuantLib::Option::arguments, 635
- QuantLib::OptionTypeFormatter, 636
- QuantLib::OrnsteinUhlenbeckProcess, 637
- QuantLib::OrnsteinUhlenbeckProcess
 - expectation, 637
 - variance, 637
- QuantLib::Oslo, 639
- QuantLib::Parameter, 640
- QuantLib::ParameterImpl, 641
- QuantLib::ParCoupon, 642
- QuantLib::ParCoupon
 - amount, 643
 - fixing, 643
 - update, 643
- QuantLib::Path, 644
- QuantLib::PathGenerator, 645
- QuantLib::PathPricer, 646
- QuantLib::Payoff, 647
- QuantLib::PercentageStrikePayoff, 648
- QuantLib::Period, 649
- QuantLib::PiecewiseConstantParameter, 650
- QuantLib::PiecewiseFlatForward, 651
- QuantLib::PiecewiseFlatForward
 - PiecewiseFlatForward, 652, 653
 - update, 653
- QuantLib::PKRCurrency, 654
- QuantLib::PlainVanillaPayoff, 655
- QuantLib::PLNCurrency, 656
- QuantLib::PoissonDistribution, 657
- QuantLib::PositiveConstraint, 658
- QuantLib::PricingEngine, 659
- QuantLib::PrimeNumbers, 660
- QuantLib::Problem, 661
- QuantLib::processing_iterator, 662
- QuantLib::PTECurrency, 664

- QuantLib::QuantoEngine, 665
- QuantLib::QuantoEngine
 - underlyingArgs, 665
- QuantLib::QuantoForwardVanillaOption, 667
- QuantLib::QuantoForwardVanillaOption
 - setupArguments, 668
- QuantLib::QuantoOptionArguments, 669
- QuantLib::QuantoOptionResults, 670
- QuantLib::QuantoTermStructure, 671
- QuantLib::QuantoTermStructure
 - today'sDate, 672
- QuantLib::QuantoVanillaOption, 673
- QuantLib::QuantoVanillaOption
 - performCalculations, 674
 - setupArguments, 674
 - setupExpired, 674
- QuantLib::Quote, 675
- QuantLib::RandomizedLDS, 676
- QuantLib::RandomizedLDS
 - nextRandomizer, 677
- QuantLib::RandomSequenceGenerator, 678
- QuantLib::RateFormatter, 679
- QuantLib::RateHelper, 680
- QuantLib::RateHelper
 - latestDate, 681
 - maturity, 681
 - setTermStructure, 681
 - update, 681
- QuantLib::Results, 682
- QuantLib::Ridder, 683
- QuantLib::Riyadh, 684
- QuantLib::ROLCurrency, 685
- QuantLib::Rounding, 686
 - Ceiling, 687
 - Closest, 687
 - Down, 687
 - Floor, 687
 - None, 687
 - Up, 687
- QuantLib::Rounding
 - Rounding, 687
 - Type, 686
- QuantLib::SalvagingAlgorithm, 688
- QuantLib::Sample, 689
- QuantLib::SARCcurrency, 690
- QuantLib::Schedule, 691
- QuantLib::Secant, 692
- QuantLib::SeedGenerator, 693
- QuantLib::SegmentIntegral, 694
- QuantLib::SEKCurrency, 695
- QuantLib::Seoul, 696
- QuantLib::SequenceFormatter, 697
- QuantLib::SequenceStatistics, 698
- QuantLib::Settings, 700
- QuantLib::Settings
 - evaluationDate, 700
 - evaluationDateGuard, 700
 - setEvaluationDate, 700
- QuantLib::SGDCurrency, 702
- QuantLib::Short, 703
- QuantLib::Short
 - amount, 703
- QuantLib::Short< ParCoupon >, 704
- QuantLib::ShortRateModel, 705
- QuantLib::ShortRateModel
 - calibrate, 706
 - update, 706
- QuantLib::ShoutCondition, 707
- QuantLib::ShoutCondition
 - applyTo, 707
- QuantLib::SimpleCashFlow, 708
- QuantLib::SimpleCashFlow
 - amount, 708
- QuantLib::SimpleDayCounter, 709
- QuantLib::SimpleQuote, 710
- QuantLib::SimpleSwap, 711
- QuantLib::SimpleSwap
 - setupArguments, 712
- QuantLib::SimpleSwap::arguments, 713
- QuantLib::SimpleSwap::results, 714
- QuantLib::Simplex, 715
- QuantLib::Simplex
 - Simplex, 715
- QuantLib::SimpsonIntegral, 716
- QuantLib::Singapore, 717
- QuantLib::SingleAssetOption, 718
- QuantLib::SingleAssetOption
 - impliedVolatility, 719
- QuantLib::Singleton, 720
- QuantLib::SITCurrency, 721
- QuantLib::SizeFormatter, 722
- QuantLib::SKKCurrency, 723
- QuantLib::SobolRsg, 724
- QuantLib::SobolRsg
 - SobolRsg, 725
- QuantLib::Solver1D, 726
- QuantLib::Solver1D
 - setMaxEvaluations, 727
 - solve, 727
- QuantLib::SquareRootProcess, 728
- QuantLib::StatsHolder, 729
- QuantLib::SteepestDescent, 730
- QuantLib::step_iterator, 731
- QuantLib::StepCondition, 732
- QuantLib::StepCondition
 - applyTo, 732
- QuantLib::stepping_iterator, 733

- QuantLib::StochasticProcess, 735
- QuantLib::StochasticProcess
 - evolve, 736
 - expectation, 736
 - update, 736
 - variance, 736
- QuantLib::StochasticProcess::discretization, 737
- QuantLib::Stock, 738
- QuantLib::Stock
 - performCalculations, 738
- QuantLib::Stockholm, 739
- QuantLib::StrikedTypePayoff, 740
- QuantLib::StringFormatter, 741
- QuantLib::StulzEngine, 742
- QuantLib::SuperSharePayoff, 743
- QuantLib::SVD, 744
- QuantLib::Swap, 745
- QuantLib::Swap
 - performCalculations, 746
 - sensitivity, 746
 - setupExpired, 746
- QuantLib::SwapRateHelper, 747
- QuantLib::SwapRateHelper
 - latestDate, 748
 - setTermStructure, 748
- QuantLib::Swaption, 749
- QuantLib::Swaption
 - setupArguments, 749
- QuantLib::Swaption::arguments, 750
- QuantLib::Swaption::results, 751
- QuantLib::SwaptionVolatilityMatrix, 752
- QuantLib::SwaptionVolatilityStructure, 753
- QuantLib::SwaptionVolatilityStructure
 - SwaptionVolatilityStructure, 754
- QuantLib::Sydney, 755
- QuantLib::SymmetricSchurDecomposition, 756
- QuantLib::SymmetricSchurDecomposition
 - SymmetricSchurDecomposition, 756
- QuantLib::Taiwan, 757
- QuantLib::TARGET, 758
- QuantLib::TermStructureConsistentModel, 759
- QuantLib::TermStructureFittingParameter, 760
- QuantLib::THBCurrency, 761
- QuantLib::Thirty360, 762
- QuantLib::Tian, 763
- QuantLib::TimeBasket, 764
- QuantLib::TimeGrid, 765
- QuantLib::TimeGrid
 - TimeGrid, 765
- QuantLib::Tokyo, 766
- QuantLib::Toronto, 768
- QuantLib::TrapezoidIntegral, 769
- QuantLib::Tree, 771
- QuantLib::TreeCapFloorEngine, 772
- QuantLib::TreeSwaptionEngine, 773
- QuantLib::TridiagonalOperator, 774
- QuantLib::TridiagonalOperator::TimeSetter, 776
- QuantLib::Trigeorgis, 777
- QuantLib::TrinomialBranching, 778
- QuantLib::TrinomialTree, 779
- QuantLib::TRLCurrency, 780
- QuantLib::TTDCurrency, 781
- QuantLib::TWDCurrency, 782
- QuantLib::TwoFactorModel, 783
- QuantLib::TwoFactorModel::ShortRateDynamics, 784
- QuantLib::TwoFactorModel::ShortRateTree, 785
- QuantLib::TypePayoff, 786
- QuantLib::UnitedKingdom, 787
 - Exchange, 788
 - Settlement, 788
- QuantLib::UnitedKingdom
 - Market, 788
- QuantLib::UnitedStates, 789
 - Exchange, 791
 - Settlement, 791
- QuantLib::UnitedStates
 - Market, 791
- QuantLib::UpFrontIndexedCoupon, 792
- QuantLib::UpRounding, 793
- QuantLib::USDCurrency, 794
- QuantLib::USDLibor, 795
- QuantLib::Value, 796
- QuantLib::VanillaOption, 797
- QuantLib::VanillaOption::engine, 798
- QuantLib::Vasicek, 799
- QuantLib::Vasicek::Dynamics, 800
- QuantLib::VEBCurrency, 801
- QuantLib::Visitor, 802
- QuantLib::VolatilityFormatter, 803
- QuantLib::Warsaw, 804
- QuantLib::WeekdayFormatter, 805
- QuantLib::Wellington, 806
- QuantLib::Xibor, 807
- QuantLib::Xibor
 - fixing, 808
 - name, 808
 - update, 808
 - Xibor, 808
- QuantLib::XiborManager, 809
- QuantLib::YieldTermStructure, 810
- QuantLib::YieldTermStructure

- compoundForward, 814
- forward, 814, 815
- forwardRate, 815
- instantaneousForward, 814
- parRate, 815
- YieldTermStructure, 813
- zeroCoupon, 813
- zeroRate, 813, 814
- zeroYield, 813
- QuantLib::ZARCurrency, 817
- QuantLib::ZARLibor, 818
- QuantLib::ZeroCurve, 819
- QuantLib::ZeroCurve
 - ZeroCurve, 820
- QuantLib::ZeroSpreadedTermStructure, 821
- QuantLib::ZeroSpreadedTermStructure
 - forwardImpl, 822
 - todayDate, 822
- QuantLib::ZeroYieldStructure, 823
- QuantLib::ZeroYieldStructure
 - compoundForwardImpl, 824
 - discountImpl, 824
 - forwardImpl, 824
 - ZeroYieldStructure, 824
- QuantLib::Zurich, 825
- Quanto option engines, 93
- Quarterly
 - datetime, 82
- rankReducedSqrt
 - QuantLib::Matrix, 550
- recalculate
 - QuantLib::LazyObject, 520
- regret
 - QuantLib::GenericRiskStatistics, 436
- RelinkableHandle
 - relinkablehandle.hpp, 1139
- relinkablehandle.hpp
 - RelinkableHandle, 1139
- removeHoliday
 - QuantLib::Calendar, 234
- reset
 - QuantLib::DiscretizedAsset, 337
 - QuantLib::DiscretizedDiscountBond, 339
 - QuantLib::DiscretizedOption, 340
- RiskStatistics
 - riskstatistics.hpp, 1005
- riskstatistics.hpp
 - RiskStatistics, 1005
- ROL
 - currencies, 79
- rollback
 - QuantLib::FiniteDifferenceModel, 395
 - QuantLib::Lattice, 516
 - QuantLib::NumericalMethod, 613
- Rounding
 - QuantLib::Rounding, 687
- SAR
 - currencies, 79
- SecondDerivative
 - QuantLib::CubicSpline, 298
- secondDerivativeAtCenter
 - valueatcenter.hpp, 923
- SEK
 - currencies, 79
- Semiannual
 - datetime, 82
- semiDeviation
 - QuantLib::GenericRiskStatistics, 435
- semiVariance
 - QuantLib::GenericRiskStatistics, 435
- sensitivity
 - QuantLib::Swap, 746
- sequencestatistics.hpp
 - DEFINE_SEQUENCE_STAT_CONST_ - METHOD_DOUBLE, 1009
 - DEFINE_SEQUENCE_STAT_CONST_ - METHOD_VOID, 1009
- setEvaluationDate
 - QuantLib::Settings, 700
- setMaxEvaluations
 - QuantLib::Solver1D, 727
- setPricingEngine
 - QuantLib::Instrument, 479
- setTermStructure
 - QuantLib::DepositRateHelper, 321
 - QuantLib::FraRateHelper, 414
 - QuantLib::RateHelper, 681
 - QuantLib::SwapRateHelper, 748
- setTime
 - QuantLib::BoundaryCondition, 218
 - QuantLib::DirichletBC, 324
 - QuantLib::NeumannBC, 601
- Settlement
 - QuantLib::Germany, 441
 - QuantLib::Italy, 501
 - QuantLib::UnitedKingdom, 788
 - QuantLib::UnitedStates, 791
- setupArguments
 - QuantLib::BarrierOption, 169
 - QuantLib::BasketOption, 175
 - QuantLib::CapFloor, 242
 - QuantLib::CliquetOption, 259
 - QuantLib::ContinuousAveraging-AsianOption, 279

- QuantLib::DiscreteAveragingAsian-Option, 331
- QuantLib::DividendVanillaOption, 343
- QuantLib::ForwardVanillaOption, 413
- QuantLib::Instrument, 479
- QuantLib::MultiAssetOption, 592
- QuantLib::OneAssetOption, 620
- QuantLib::OneAssetStrikedOption, 624
- QuantLib::QuantoForwardVanilla-Option, 668
- QuantLib::QuantoVanillaOption, 674
- QuantLib::SimpleSwap, 712
- QuantLib::Swaption, 749
- setupExpired
 - QuantLib::Instrument, 479
 - QuantLib::MultiAssetOption, 592
 - QuantLib::OneAssetOption, 620
 - QuantLib::QuantoVanillaOption, 674
 - QuantLib::Swap, 746
- SGD
 - currencies, 79
- Short-rate modelling framework, 102
- shortfall
 - QuantLib::GenericRiskStatistics, 436
- shortfloatingcoupon.hpp
 - ShortFloatingRateCoupon, 871
- ShortFloatingRateCoupon
 - shortfloatingcoupon.hpp, 871
- Side
 - QuantLib::BoundaryCondition, 217
- Simplex
 - QuantLib::Simplex, 715
- SIT
 - currencies, 79
- skewness
 - QuantLib::GeneralStatistics, 431
 - QuantLib::IncrementalStatistics, 472
- SKK
 - currencies, 79
- SobolRsg
 - QuantLib::SobolRsg, 725
- solve
 - QuantLib::Solver1D, 727
- standardDeviation
 - QuantLib::GeneralStatistics, 431
 - QuantLib::IncrementalStatistics, 471
- standardDeviations
 - QuantLib::CovarianceDecomposition, 290
- Statistics
 - statistics.hpp, 1012
- statistics.hpp
 - Statistics, 1012
- String functions, 128
- Swaption engines, 94
- SwaptionVolatilityStructure
 - QuantLib::SwaptionVolatilityStructure, 754
- SymmetricSchurDecomposition
 - QuantLib::SymmetricSchur-Decomposition, 756
- targetValueAndGradient
 - QuantLib::LeastSquareProblem, 522
- Template capabilities, 131
- templateMacros
 - QL_TYPENAME, 131
- Term structures, 120
- TermStructure
 - termstructure.hpp, 1165
- termstructure.hpp
 - TermStructure, 1165
- THB
 - currencies, 79
- Time functions, 127
- TimeGrid
 - QuantLib::TimeGrid, 765
- today'sDate
 - QuantLib::BaseTermStructure, 173
 - QuantLib::DriftTermStructure, 353
 - QuantLib::ForwardSpreadedTerm-Structure, 411
 - QuantLib::QuantoTermStructure, 672
 - QuantLib::ZeroSpreadedTerm-Structure, 822
- topPercentile
 - QuantLib::GeneralStatistics, 432
- TRL
 - currencies, 79
- TTD
 - currencies, 79
- TWD
 - currencies, 79
- Type
 - QuantLib::ExchangeRate, 371
 - QuantLib::Rounding, 686
- Unadjusted
 - datetime, 81
- underlyingArgs
 - QuantLib::QuantoEngine, 665
- unfreeze
 - QuantLib::LazyObject, 520
- Up
 - QuantLib::Rounding, 687
- update
 - QuantLib::AffineTermStructure, 140
 - QuantLib::BaseTermStructure, 173

- QuantLib::BlackModel, 197
- QuantLib::BlackScholesProcess, 201
- QuantLib::CalibrationHelper, 239
- QuantLib::CapVolatilityVector, 251
- QuantLib::CompositeQuote, 270
- QuantLib::DerivedQuote, 323
- QuantLib::ExtendedDiscountCurve, 380
- QuantLib::GenericModelEngine, 434
- QuantLib::IndexedCoupon, 475
- QuantLib::LatticeShortRateModel-Engine, 518
- QuantLib::LazyObject, 519
- QuantLib::Observer, 618
- QuantLib::ParCoupon, 643
- QuantLib::PiecewiseFlatForward, 653
- QuantLib::RateHelper, 681
- QuantLib::ShortRateModel, 706
- QuantLib::StochasticProcess, 736
- QuantLib::Xibor, 808
- USD
 - currencies, 79
- Utilities, 121
- valueAtCenter
 - valueatcenter.hpp, 923
- valueatcenter.hpp
 - firstDerivativeAtCenter, 923
 - secondDerivativeAtCenter, 923
 - valueAtCenter, 923
- valueAtRisk
 - QuantLib::GenericRiskStatistics, 436
- Vanilla option engines, 95
- variance
 - QuantLib::EulerDiscretization, 364
 - QuantLib::GeneralStatistics, 431
 - QuantLib::IncrementalStatistics, 471
 - QuantLib::OrnsteinUhlenbeckProcess, 637
 - QuantLib::StochasticProcess, 736
- variances
 - QuantLib::CovarianceDecomposition, 290
- VEB
 - currencies, 79
- Weekday
 - datetime, 82
- Xetra
 - QuantLib::Germany, 441
- Xibor
 - QuantLib::Xibor, 808
- yield
 - QuantLib::Bond, 215
 - YieldTermStructure
 - QuantLib::YieldTermStructure, 813
 - ZAR
 - currencies, 79
 - zeroCoupon
 - QuantLib::YieldTermStructure, 813
 - ZeroCurve
 - QuantLib::ZeroCurve, 820
 - zeroRate
 - QuantLib::YieldTermStructure, 813, 814
 - zeroYield
 - QuantLib::YieldTermStructure, 813
 - zeroYieldImpl
 - QuantLib::CompoundForward, 272
 - QuantLib::DiscountStructure, 328
 - QuantLib::ForwardRateStructure, 409
 - QuantLib::ForwardSpreadedTerm-Structure, 411
 - ZeroYieldStructure
 - QuantLib::ZeroYieldStructure, 824